Python Programming

ایک مکمل اردو رہنما

Dr. Muhammad Siddique 03006329919

Dr. Muhammad Siddique

HOD Artificial intelligence, NFC IET Multan

Contact Information

Engr. Dr. Muhammad Siddique

HOD, Department of Artificial Intelligence
NFC Institute of Engineering & Technology, Multan

Email: siddique9171@gmail.com

Cell: +92-300-6329919

Dr. Muhammad Siddique 03006329919

For online Python learning, guidance, or consultation, feel free to get in touch.

Students and professionals are warmly welcome to contact for online sessions on Python programming, from beginner to advanced levels.

© 2024 Engr. Dr. Muhammad Siddique | All Rights Reserved.

Dr. Muhammad Siddique 03006329919

Contents

1 Programming Environment Setup 3

2 Data Types and Variables 11

11 . . . . . variables 1.0.2
12 . . . . . variables اور سادہ ڈیٹا اقسام 2.0.2
12 . . . . . variables کے ساتھ نام کی غلطیاں 3.0.2
12 . . . . . variables: لیبلز کی طرح 4.0.2
12 . . . . . ٹیبلز اور نیو لائنز کے ساتھ سفید جگہ شامل کرنا 5.0.2
13 . . . . . variables اور سادہ ڈیٹا کی اقسام 1.2
14 . . . . . code 2.2
41 . . . . . List کمپریہنشنز 3.2
56 . . . . . if-Statements 4.2
57 . . . . . Statements-if-Simple 1.4.2
58 . . . . . Statements-if-else 2.4.2
58 . . . . . if-elif-else چین 5.2
59 . . . . . if-elif-else بلاکس کا استعمال 6.2
60 . . . . . else بلاک کو ختم کرنا 7.2
60 . . . . . if-elif-else 8.2
61 . . . . . List کے ساتھ statement-if کا استعمال 9.2
61 . . . . . خصوصی elements کی جانچ 10.2
62 . . . . . یہ جانچنا کہ Empty-List نہیں ہے 11.2
63 . . . . . Lists-Multiple کا استعمال 12.2
63 . . . . . Excercise 13.2
65 . . . . . statement-IF کو اسٹائل کریں 14.2
65 . . . . . Excercise 15.2
66 . . . . . خلاصہ 16.2
75 . . . . . ڈکشنری کی keys مخصوص ترتیب میں Loop کرنا 17.2
75 . . . . . ڈکشنری میں تمام values کے ذریعے Loop کرنا 18.2
77 . . . . . Nesting 19.2
77 . . . . . ڈکشنریوں کی List 1.19.2
78 . . . . . Aliens کے ساتھ کام کرنا 2.19.2
80 . . . . . ایک Dictionary میں List 20.2
81 . . . . . Dictionary میں Dictionary 21.2

Dr. Muhammad Siddique 03006329916

22.2 inputUser اور Loop-While 83
23.2 input() فنکشن کیسے کام کرتا ہے 83
24.2 واضح پرامپٹس لکھنا 84
25.2 int() کا استعمال کر کے عددی input حاصل کرنا 85
26.2 int() فنکشن کا استعمال ایک حقیقی پروگرام میں 86
27.2 موڈیولو آپریٹر 86
28.2 While-Loop 87
29.2 Loop-While کا عمل میں آنا 87
30.2 User کو جب چاہے پروگرام بند کرنے کی اجازت دینا 88
31.2 پرچم (Flag) کا استعمال 89
32.2 break-statement 90
33.2 loop-میں-continue-statement 91
34.2 loop-infinite سے بچنا 91
35.2 Loop-While کے ساتھ Lists اور Dictionaries کا استعمال 92
36.2 ایک لسٹ سے دوسری لسٹ میں elements منتقل کرنا 93
37.2 ایک لسٹ سے مخصوص values کو ہٹانا 93
38.2 InputUser کے ساتھ ڈکشنری بھرنا 94

Dr. Muhammad Siddique 03006329919

Python in Urdu

Engr. Dr. Muhammad Siddique
NFC Institute of Engineering
Technology, Multan

October 2024

Dr. Muhammad Siddique 0300652919

Dr. Muhammad Siddique 03006329919

Chapter 1

Programming Environment Setup

یہ باب پائتھن پروگرامنگ کے عملی آغاز سے پہلے سب سے اہم مرحلے یعنی پروگرامنگ ماحول(EnvironmentProgramming) کی مکمل وضاحت اور ترتیب پر مشتمل ہے۔پروگرامنگ سیکھنے کے لیے ضروری ہے کہ آپ کے کمپیوٹر پر تمام مطلوبہ سافٹ ویئر درست طریقے سے انسٹال اور ترتیب دیا گیا ہو تاکہ آپ بآسانی code لکھ سکیں، چلا سکیں، اور ڈی بگ کر سکیں۔

پروگرامنگ ماحول کیا ہے؟

Dr. Muhammad Siddique 03006329919

پروگرامنگ ماحول(Programming-Environment) دراصل وہ تمام سافٹ ویئر، ٹولز، سیٹنگز اور فائلز کا مجموعہ ہوتا ہے جو کسی مخصوص پروگرامنگ زبان میں code لکھنے، چلانے اور جانچنے کے لیے استعمال ہوتے ہیں۔اس میں درج ذیل شامل ہو سکتے ہیں:

- پروگرامنگ زبان کا انٹرپریٹر یا کمپائلر (مثلاً Python-interpreter)
- code لکھنے والا ایڈیٹر –یا– IDE (مثلاً Visual-Studio-Code)
- اضافی پلگ انز –یا– ایکسٹینشنز (Python-extension-for-VS-Code)
- سسٹم سیٹنگز جیسے PATH-environment-variable

پروگرامنگ ماحول کو درست طور پر ترتیب دینا نئے پروگرامرز کے لیے مشکل ہو سکتا ہے، لیکن اس باب میں ہم قدم بہ قدم ان مراحل کو انتہائی آسان انداز میں statement-کریں گے تاکہ آپ بغیر کسی رکاوٹ کے پائتھن پروگرامنگ سیکھ سکیں۔

پائتھن کیا ہے؟

پائتھن ایک ہائی لیول، انٹرپریٹیڈ پروگرامنگ زبان ہے جو سادگی، پڑھنے میں آسانی، اور ورسٹائل نیچر کی وجہ سے بے حد مقبول ہے۔پائتھن کا سادہ Syntax اسے نئے سیکھنے والوں کے لیے بہترین بناتا ہے۔

پائتھن میں پروگرام لکھنے کے لیے سب سے پہلے اس کی انسٹالیشن اور ایک اچھا code ایڈیٹر ہونا ضروری ہے۔

پائتھن انسٹال کرنا

Windows پر Python انسٹال کریں

1. سب سے پہلے https://www.python.org/downloads/ پر جائیں۔
2. وہاں سے Python کا تازہ ترین ورژن ڈاؤن لوڈ کریں (مثلاً Python3.11.x)۔
3. انسٹالر کو چلائیں۔
4. PATHtoPythonAdd کے آپشن کو ضرور چیک کریں—یہ ایک اہم قدم ہے۔
5. پھر Install"" Now" پر کلک کریں۔
6. انسٹالیشن مکمل ہونے کے بعد آپ Command-Prompt کھول کر python --version لکھیں تاکہ تصدیق ہو سکے کہ Python کامیابی سے انسٹال ہو چکی ہے۔

PATH Variable کیا ہے؟

جب آپ Python انسٹال کرتے ہیں، تو "PATH" ایک سسٹم سیٹنگ ہوتی ہے جو کمپیوٹر کو بتاتی ہے کہ Python کہاں انسٹال ہے۔ اگر PATH سیٹ نہ ہو تو آپ کو ہر بار Python چلانے کے لیے مکمل address دینا پڑے گا۔

Dr. Muhammad Siddique 03006329919

Python-Shell استعمال کریں

Python انسٹال کرنے کے بعد، آپ Python Shell میں جا کر فوری code آزما سکتے ہیں۔

- Start Menu میں جا کر "Python" لکھیں اور Python 3.x پر کلک کریں۔
- آپ کو >>> پرامپٹ نظر آئے گا۔
- اب آپ وہاں یہ code لکھیں:

```
print("Python successfully installed!")
```

اگر output میں یہی لائن پرنٹ ہو تو اس کا مطلب ہے کہ Python کامیابی سے انسٹال ہو چکی ہے۔

VS-Code انسٹال کرنا

Visual Studio Code یا VS Code ایک مقبول اور طاقتور code ایڈیٹر ہے۔ یہ Microsoft کی جانب سے تیار کیا گیا ہے اور Python سمیت درجنوں زبانوں کو سپورٹ کرتا ہے۔

VS-Code ڈاؤن لوڈ اور انسٹال کریں

1. https://code.visualstudio.com/ پر جائیں۔
2. Windows کے لیے Installer ڈاؤن لوڈ کریں۔
3. Setup فائل چلائیں اور SettingsDefault کے ساتھ انسٹال کریں۔
4. انسٹالیشن مکمل ہونے کے بعد CodeVS کو لانچ کریں۔

Python-Extension انسٹال کریں

CodeVS میں Python-code صحیح طور پر چلانے کے لیے ExtensionPython انسٹال کرنا ضروری ہے:

- CodeVS میں Extensions پین کھولیں (بائیں جانب square آئیکون پر کلک کریں)۔
- Search بار میں Python لکھیں۔
- Microsoft کی ExtensionPython کو انسٹال کریں۔

Dr. Muhammad Siddique 03006329919

CodeVS میں Python پروگرام کیسے چلائیں؟

1. CodeVS کھولیں۔
2. ایک نئی فائل بنائیں اور hello.py کے نام سے محفوظ کریں۔
3. درج ذیل code لکھیں:

```
print("Welcome to Python in VS Code!")
```

1. Shift+Right Click کریں اور Run "Python File in" Terminal کو منتخب کریں۔
2. آپ کو ٹرمینل میں output نظر آئے گی۔

Best-Practices

- ہر Python فائل کا نام چھوٹے حروف میں رکھیں اور اسپیس کی جگہ _ کا استعمال کریں۔
- Python پروگرامز کے لیے الگ فولڈر بنائیں۔
- ورچوئل انوائرمنٹ (virtual environment) استعمال کرنا سیکھیں (آئندہ باب میں تفصیل سے statement- ہوگا)۔

Troubleshooting

- اگر python کمانڈ کام نہ کرے، تو Command-Prompt بند کر کے دوبارہ کھولیں یا Python انسٹالر سے Modify کر کے PATH دوبارہ شامل کریں۔
- VS-Code میں اگر Run کا بٹن نہ آئے تو Python-Extension چیک کریں۔

Windows:

```
> env\Scripts\activate
```

- فعال ہونے پر آپ کو ٹرمینل میں '(env)' جیسا کچھ نظر آئے گا۔
- اب جو بھی پیکج انسٹال ہو گا وہ اسی environment میں ہو گا۔

Deactivation

Virtual-environment کو بند کرنے کے لیے صرف یہ کمانڈ دیں:

```
$ deactivate
```

Dr. Muhammad Siddique 03006329919

Python پیکجز اور pip

Python میں اضافی فیچرز اور لائبریریز کو پیکجز کہا جاتا ہے، جنہیں pip کی مدد سے انسٹال کیا جاتا ہے۔

pip کیا ہے؟

pip پیکج مینیجر ہے Python کا، جو لائبریریز کو انسٹال، اپڈیٹ، اور منیج کرنے کے لیے استعمال ہوتا ہے۔

pip استعمال کریں

```
$ pip install numpy
```

انسٹال شدہ پیکجز دیکھیں

```
$ pip list
```

پیکج کو اَن انسٹال کریں

```
$ pip uninstall numpy
```

VS-Codeایکسٹینشنز کی تفصیل

CodeVSکی طاقت اس کے ایکسٹینشنز میں ہے، جن سے نگ codeمزید آسان اور موثر ہو جاتی ہے۔

- code:ExtensionPythonچلانے،syntax،highlightingاورIntelliSenseکے لیے ضروری۔
- Pylintیاcode:Flake8کی خامیاں ظاہر کرنے کے لیے۔
- NotebooksJupyter:Jupyterکے ساتھ کام کرنے کے لیے۔
- Alt+Ctrl+N:RunnerCodeسے codeچلانے کی سہولت دیتا ہے۔
- controlversionGit:GitLensکے لیے۔

Easily-Install-Extensions

- CodeVSمیں بائیں طرف Extensionsآئیکون پر کلک کریں۔
- سرچ بار میں نام لکھیں(مثلاً:Python)
- Installبٹن پر کلک کریں۔

Dr. Muhammad Siddique 03006329019

VS-Code-Shortcuts

- ` + Ctrl ٹرمینل کھولیں
- B + Ctrl سائڈ بار ہائیڈ/شو
- P + Ctrl فائل سرچ
- P + Shfit + Ctrl کمانڈ پیلیٹ

پائتھن میںVirtual-Environmentبنانا

Virtual environmentایک ایسا isolatedماحول ہوتا ہے جس میں آپ مخصوص پراجیکٹس کے لیے Pythonپیکجز انسٹال کر سکتے ہیں، بغیر سسٹم Pythonکو متاثر کیے۔

Virtualenvکیوں ضروری ہے؟

- ہر پراجیکٹ میں مختلف لائبریریز اور ورژنز کی ضرورت ہو سکتی ہے۔
- سسٹم Pythonکو خراب کیے بغیر آپ ہر پراجیکٹ کے لیے ایک علیحدہ ماحول بنا سکتے ہیں۔

Virtual-Environment بنائیں

1. Terminal یا CodeVS کھولیں۔
2. نئی ڈائریکٹری بنائیں: mkdir myproject
3. اس میں جائیں: cd myproject
4. Virtual environment بنائیں:

```
$ python -m venv env
```

Virtual-Environment ایکٹیویٹ کریں

پائتھن انسٹالیشن کے عام مسائل اور ان کا حل

نئے users کو پائتھن اور وی ایس code کی انسٹالیشن کے دوران کئی مشکلات پیش آسکتی ہیں۔ یہاں ہم عام مسائل اور ان کے حل statement-کریں گے۔

Dr. Muhammad Siddique 03006329919

Python کمانڈ نہیں چل رہی؟

- اگر آپ نے کمانڈ پرامپٹ یا ٹرمینل میں python لکھا اور "command not found" یا "is not recognized" کا پیغام آیا تو اس کا مطلب ہے کہ Python کا PATH درست طریقے سے سیٹ نہیں ہوا۔
- حل:
  1. Python کو دوبارہ انسٹال کریں۔
  2. انسٹالیشن کے دوران "Add Python to PATH" کے باکس کو ضرور منتخب کریں۔
  3. یا "Advanced Options" میں جا کر "Add to environment variables" کو چیک کریں۔

CodeVS میں Run کا بٹن غائب ہے؟

- CodeVS میں اگر Run کا بٹن نظر نہ آئے، تو سب سے پہلے چیک کریں کہ Python extension انسٹال ہے یا نہیں۔
- اگر انسٹال ہے، تو CodeVS کو بند کر کے دوبارہ کھولیں۔
- بعض اوقات restart یا system reboot کرنے سے مسئلہ حل ہو جاتا ہے۔

Python-interpreter سیلیکٹ نہیں ہو رہا؟

- CodeVS میں Ctrl+Shift+P دبائیں۔
- "Python: Select Interpreter" لکھ کر enter کریں۔
- اپنے virtual environment یا سسٹم Python کو منتخب کریں۔

پہلا مکمل Python پراجیکٹ

اب ہم پائتھن کا ایک مکمل پراجیکٹ بنائیں گے تاکہ آپ کو پروگرامنگ کا عملی تجربہ حاصل ہو۔

Step-by-Step پراجیکٹ گائیڈ

- فولڈر بنائیں: my_first_project
- اس میں ایک فائل بنائیں: calculator.py
- درج ذیل code لکھیں:

```
# Simple Calculator in Python

def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        return "Division by zero error!"
    return a / b

print("Select operation:")
print("1. Add")
print("2. Subtract")
print("3. Multiply")
print("4. Divide")

choice = input("Enter choice (1/2/3/4): ")

a = float(input("Enter first number: "))
b = float(input("Enter second number: "))

if choice == '1':
    print("Result:", add(a, b))
elif choice == '2':
    print("Result:", subtract(a, b))
elif choice == '3':
    print("Result:", multiply(a, b))
elif choice == '4':
    print("Result:", divide(a, b))
else:
    print("Invalid input")
```

Dr. Muhammad Siddique 03006329919

پراجیکٹ کے نکات

- یہ پروگرام چار آپریشنز پر مشتمل ہے: جمع، تفریق، ضرب، اور تقسیم۔
- input() کے ذریعے یوزر سے ڈیٹا لیا گیا ہے۔
- اگر صفر پر تقسیم کی کوشش ہو تو خاص Error میسج ظاہر ہوتا ہے۔
- یہ پراجیکٹ Python کی بنیاد پر مبنی ہے اور نئے سیکھنے والوں کے لیے بہترین ہے۔

یاد رکھنے کے نکات

- code کو مرحلہ وار لکھیں اور ہر قدم پر اسے چلائیں۔
- غلطی ہو تو Error میسج کو غور سے پڑھیں — وہ اکثر حل کا سرا دے دیتا ہے۔
- پراجیکٹ کو فولڈر میں منظم رکھیں۔
- فائلز کو واضح نام دیں جیسے: hello.py، calculator.py۔
- پائتھن کی آفیشل ڈاکیومنٹیشن https://docs.python.org/3 سے رہنمائی حاصل کریں۔

Dr. Muhammad Siddique 03006312919

اگلا مرحلہ

اب جب کہ آپ نے پائتھن اور CodeVS انسٹال کر لیا ہے، اور ایک سادہ پراجیکٹ بھی بنا لیا ہے، تو اگلے باب میں ہم Python کے بنیادی سینٹیکس، ڈیٹا ٹائپس، اور variables کا تفصیلی جائزہ لیں گے۔

پائتھن کی دنیا میں خوش آمدید!

Chapter 2

Data Types and Variables

جب آپ hello_world.py فائل کو چلائیں گے، تو .py کا اختتام اس بات کی نشاندہی کرتا ہے کہ یہ ایک Python پروگرام ہے۔ آپ کا ایڈیٹر پھر اس فائل کو Python انٹرپریٹر کے ذریعے چلاتا ہے، جو پروگرام کو پڑھتا ہے اور یہ تعین کرتا ہے کہ پروگرام کے ہر word- کا کیا مطلب ہے۔ مثال کے طور پر، جب انٹرپریٹر print کو قوسین کے ساتھ دیکھتا ہے، تو یہ اس میں موجود مواد کو سکرین پر پرنٹ کرتا ہے۔

جب آپ اپنے پروگرام لکھتے ہیں تو آپ کا ایڈیٹر پروگرام کے مختلف حصوں کو مختلف طریقوں سے ہائی لائٹ کرتا ہے۔ مثلاً، یہ پہچانتا ہے کہ print () ایک فنکشن کا نام ہے اور اسے ایک رنگ میں دکھاتا ہے۔ یہ پہچانتا ہے کہ "Hello Python world!" Python کا code نہیں ہے اور اسے ایک دوسرے رنگ میں دکھاتا ہے۔ اس فیچر کو سینٹیکس ہائی لائٹنگ کہا جاتا ہے اور جب آپ اپنے پروگرام لکھنا شروع کرتے ہیں تو یہ بہت مفید ہوتا ہے۔

Dr. Muhammad Siddique 03062299178

1.0.2 variables

آیئے hello_world.py میں ایک variable استعمال کرنے کی کوشش کرتے ہیں۔ فائل کے آغاز میں ایک نئی لائن شامل کریں اور دوسری لائن کو تبدیل کریں:

```
hello_world.py
message = "Hello Python world!"
print(message)
```

اس پروگرام کو چلائیں اور دیکھیں کیا ہوتا ہے۔ آپ کو وہی output نظر آنا چاہیے جو آپ نے پہلے دیکھا تھا:

```
Hello Python world!
```

ہم نے ایک variable کو message شامل کیا ہے۔ ہر variable کسی value- سے جڑا ہوتا ہے، جو اس variable کے ساتھ منسلک معلومات ہوتی ہے۔ اس صورت میں value- "Hello Python world!" ہے۔ variable شامل کرنے سے Python انٹرپریٹر کے لیے تھوڑا سا مزید کام ہوتا ہے۔ جب وہ پہلی لائن کو پروسیس کرتا ہے، تو یہ message variable کو "Hello Python world!" کی value- سے جوڑ دیتا ہے۔ جب وہ دوسری لائن پر پہنچتا ہے، تو یہ message کے ساتھ جڑی value- کو سکرین پر پرنٹ کرتا ہے۔

2.0.2 variables اور سادہ ڈیٹا اقسام

Python میں variables استعمال کرتے وقت آپ کو کچھ قواعد اور رہنمائیوں پر عمل کرنا ضروری ہے۔ کچھ قواعد کی خلاف ورزی کرنے سے آپ کو ایررز ملیں گے؛ جبکہ دیگر رہنمائیاں آپ کے code کو پڑھنے اور سمجھنے میں آسانی فراہم کرتی ہیں۔ آپ کو variables استعمال کرتے وقت درج ذیل قواعد کو یاد رکھنا چاہیے:

- variable نام صرف حروف، اعداد، اور انڈر سکورز پر مشتمل ہو سکتے ہیں۔ وہ ایک حرف یا انڈر سکور سے شروع ہو سکتے ہیں، لیکن نمبر سے شروع نہیں ہو سکتے۔
- variable ناموں میں اسپیسز کی اجازت نہیں ہے، لیکن آپ word-کو الگ کرنے کے لیے انڈر سکور استعمال کر سکتے ہیں۔
- Python کی محفوظ کی گئی کی ورڈز اور فنکشن ناموں کو variable نام کے طور پر استعمال نہ کریں۔
- variable نام مختصر لیکن وضاحت دینے والے ہونے چاہئیں۔

Dr. Muhammad Siddique 03006329919

3.0.2 variables کے ساتھ نام کی غلطیاں

ہر پروگرامر غلطیاں کرتا ہے، اور زیادہ تر روزانہ غلطیاں کرتے ہیں۔ اچھی پروگرامنگ کے لیے غلطیوں کو مؤثر طریقے سے ٹھیک کرنا آنا چاہیے۔ یہاں ایک غلطی ہے جو آپ کے پروگرام میں ہو سکتی ہے:

```
message = "Hello Python Crash Course reader!"
print(mesage)   % اسپیلنگ غلط
```

جب آپ کے پروگرام میں کوئی ایرر آتی ہے، تو Python انٹرپریٹر آپ کی مدد کے لیے ایک ٹریک بیک فراہم کرتا ہے تاکہ آپ کو یہ سمجھنے میں مدد ملے کہ کہاں غلطی ہو رہی ہے۔

4.0.2 variables: لیبلز کی طرح

variables کو اکثر "ڈبے" کے طور پر statement-کیا جاتا ہے جہاں آپ value-میں store کر سکتے ہیں۔ لیکن یہ حقیقت میں درست نہیں ہے کہ variables صرف ایک باکس کی طرح ہوتے ہیں۔ آپ انہیں زیادہ صحیح طور پر ایک لیبل کہہ سکتے ہیں جسے آپ کسی value-سے جوڑ سکتے ہیں۔

5.0.2 ٹیبلز اور نیو لائنز کے ساتھ سفید جگہ شامل کرنا

پروگرامنگ میں سفید جگہ سے مراد کوئی غیر پرنٹنگ کیریکٹرز جیسے اسپیسز، ٹیبلز، اور اینڈ آف لائن سمبلز ہیں۔ آپ اپنے output کو بہتر بنانے کے لیے سفید جگہ کا استعمال کر سکتے ہیں۔

```
print("Python")
```

اسے آپ تب اور نیو لائن کے ساتھ اس طرح کر سکتے ہیں:

```
print("Python\tis fun")
```

1.2 variables اور سادہ ڈیٹا کی اقسام

اس باب میں آپ ان مختلف اقسام کے ڈیٹا کے بارے میں سیکھیں گے جن کے ساتھ آپ اپنے پائتھن پروگرامز میں کام کر سکتے ہیں۔ آپ یہ بھی سیکھیں گے کہ variables کو اپنے پروگرامز میں ڈیٹا کی نمائندگی کے لئے کیسے استعمال کیا جاتا ہے۔

جب آپ hello_world.py چلاتے ہیں تو حقیقت میں کیا ہوتا ہے

آیئے دیکھتے ہیں کہ جب آپ hello_world.py چلاتے ہیں تو پائتھن کیا کرتا ہے۔ حقیقت یہ ہے کہ پائتھن کافی کام کرتا ہے، یہاں تک کہ جب ایک سادہ پروگرام چلایا جاتا ہے:

Dr. Muhammad Siddique 03006329919

```
hello_world.py
print("Hello Python world!")
```

جب آپ یہ code چلاتے ہیں، تو آپ کو درج ذیل output نظر آنی چاہیے:

```
Hello Python world!
```

جب آپ فائل hello_world.py چلاتے ہیں، تو .py کا اختتام اس بات کی نشاندہی کرتا ہے کہ یہ فائل ایک پائتھن پروگرام ہے۔ آپ کا ایڈیٹر فائل کو پائتھن انٹرپریٹر کے ذریعے چلاتا ہے، جو پروگرام کو پڑھتا ہے اور اس کے ہر word- کا مطلب سمجھتا ہے۔ مثلاً، جب انٹرپریٹر print دیکھتا ہے جو بریکٹ کے ساتھ ہوتا ہے، تو یہ اس کے اندر موجود مواد کو اسکرین پر پرنٹ کر دیتا ہے۔

variables

آیئے hello_world.py میں ایک variable استعمال کرتے ہیں۔ فائل کے شروع میں ایک نیا لائن شامل کریں، اور دوسرے لائن میں ترمیم کریں:

```
hello_world.py
message = "Hello Python world!"
print(message)
```

یہ پروگرام چلائیں تاکہ دیکھ سکیں کہ کیا ہوتا ہے۔ آپ کو وہی output نظر آنی چاہیے جو آپ نے پہلے دیکھا تھا:

```
Hello Python world!
```

ہم نے ایک variable شامل کیا جسے message کہا جاتا ہے۔ ہر variable کسی value سے منسلک ہوتا ہے، جو اس variable سے وابستہ معلومات ہے۔ اس معاملے میں value "Hello Python" متن ہے۔

variables کے نام رکھنے اور استعمال کرنے کے اصول

جب آپ پائتھن میں variables استعمال کرتے ہیں تو آپ کو کچھ اصولوں اور ہدایات پر عمل کرنا ہوتا ہے:

- variable کے نام صرف حروف، نمبروں، اور انڈر سکور پر مشتمل ہو سکتے ہیں۔
- variable کے نام میں Empty جگہ کی اجازت نہیں ہے۔
- پائتھن کے -reserved--word- اور فنکشن کے نام variable کے طور پر استعمال نہ کریں۔
- variable کے نام مختصر لیکن وضاحتی ہونے چاہئیں۔

Dr. Muhammad Siddique 03006329919

code 2.2

یہاں ایک مثال دی گئی ہے کہ جب آپ غلطی سے variable کا نام غلط لکھتے ہیں تو پائتھن کیا ٹریس بیک فراہم کرتا ہے:

```
1 Traceback (most recent call last):
2   File "hello_world.py", line 2, in <module>
3     print(mesage)
4           ^^^^^^
5 NameError: name 'mesage' is not defined. Did you mean: 'message'?
```

output بتاتا ہے کہ فائل hello-world.py کی لائن 2 میں ایک غلطی واقع ہوئی ہے۔ انٹرپریٹر اس لائن کو دکھاتا ہے تاکہ ہم جلدی سے مسئلے کو تلاش کر سکیں اور یہ بھی بتاتا ہے کہ اسے کس قسم کی غلطی ملی ہے۔ اس معاملے میں، اسے ایک NameError ملا ہے اور رپورٹ کرتا ہے کہ جو mesage-variable پرنٹ ہو رہا ہے، وہ ڈیفائن نہیں کیا گیا۔ پائتھن فراہم کردہ variable کے نام کو نہیں پہچان سکا۔ ایک NameError عام طور پر یہ بتاتا ہے کہ ہم نے یا تو variable کی value- کو استعمال کرنے سے پہلے مقرر کرنا بھول گئے ہیں، یا ہم نے variable کے نام کو داخل کرتے وقت ہجے کی غلطی کی ہے۔ اگر پائتھن کو ایک ایسا variable نام ملتا ہے جو غیر پہچانے گئے نام کے مشابہ ہو، تو یہ پوچھتا ہے کہ آیا یہ وہی نام ہے جو آپ استعمال کرنا چاہتے تھے۔

اس مثال میں ہم نے message-variable کے نام میں دوسرے لائن میں ایک حرف s چھوڑ دیا ہے۔ پائتھن انٹرپریٹر آپ کے code کی ہجے چیک نہیں کرتا، لیکن یہ اس بات کو یقینی بناتا ہے کہ variables کے نام مستقل طور پر ہجے کیے گئے ہوں۔ مثلاً، دیکھیں کیا ہوتا ہے جب ہم variable کی declare والی لائن میں message کو غلط ہجے کریں:

```
mesage = "Hello Python Crash Course reader!"
print(mesage)
```

اس معاملے میں، پروگرام کامیابی سے چلتا ہے!

```
Hello Python Crash Course reader!
```

variable کے نام ملتے ہیں، اس لیے پائتھن کو کوئی مسئلہ نظر نہیں آتا۔ پروگرامنگ زبانیں سخت ہیں، لیکن یہ اچھی اور بری ہجے کو نظرانداز کرتی ہیں۔ نتیجتاً، آپ کو variables کے نام بنانے اور code لکھتے وقت انگریزی ہجے اور گرامر کے قواعد کو مدنظر رکھنے کی ضرورت نہیں ہوتی۔

بہت سی پروگرامنگ کی غلطیاں سادہ، ایک حرفی ٹائپوز ہوتی ہیں جو پروگرام کی کسی ایک لائن میں ہوتی ہیں۔ اگر آپ خود کو ان میں سے کسی غلطی کو تلاش کرنے میں زیادہ وقت لگاتے ہوئے پائیں، تو جان لیں کہ آپ اچھی کمپنی میں ہیں۔ بہت سے تجربہ کار اور باصلاحیت پروگرامرز ان قسم کی چھوٹی غلطیوں کو تلاش کرنے میں گھنٹوں صرف کرتے ہیں۔ اسے ہنس کر برداشت کریں اور آگے بڑھیں، یہ جانتے ہوئے کہ یہ آپ کی پروگرامنگ زندگی میں بار بار ہوگا۔

variable-are-Labels

Dr. Muhammad Siddique 03006329919

variables کو اکثر ڈبے کے طور پر statement- کیا جاتا ہے جس میں آپ values کو store کر سکتے ہیں۔ یہ خیال آپ کے لیے اس وقت مددگار ہو سکتا ہے جب آپ variables پہلی بار استعمال کرتے ہیں، لیکن یہ اس بات کا درست طریقہ نہیں ہے کہ پائتھن کے اندر variables کی نمائندگی کیسے ہوتی ہے۔ یہ سوچنا زیادہ بہتر ہے کہ variables لیبلز ہیں جو آپ values کو assign کر سکتے ہیں۔ آپ یہ بھی کہہ سکتے ہیں کہ ایک variable کسی خاص value کا حوالہ دیتا ہے۔

یہ فرق شاید آپ کے ابتدائی پروگرامز میں زیادہ اہم نہ ہو، لیکن اسے پہلے سیکھنا زیادہ بہتر ہے۔ کسی موقع پر، آپ کو variable کے غیر متوقع برتاؤ کا سامنا ہوگا، اور variables کے کام کرنے کے طریقے کی درست understanding آپ کو اپنے code میں ہونے والے مسائل کو سمجھنے میں مدد دے گی۔

نوٹ: نئے پروگرامنگ تصورات کو سمجھنے کا بہترین طریقہ یہ ہے کہ انہیں اپنے پروگرامز میں آزمایا جائے۔ اگر آپ اس کتاب کی کسی excercise پر کام کرتے ہوئے پھنس جائیں، تو تھوڑی دیر کے لیے کچھ اور کرنے کی کوشش کریں۔ اگر آپ اب بھی پھنسے ہوئے ہیں، تو متعلقہ باب کے حصے کا جائزہ لیں۔ اگر آپ کو اب بھی مدد کی ضرورت ہے، تو Appendix C میں دی گئی تجاویز دیکھیں۔

خود آزمائیے

ہر ایک excercise کو پورا کرنے کے لیے ایک علیحدہ پروگرام لکھیں۔ ہر پروگرام کو ایک فائل نام کے ساتھ محفوظ کریں جو معیاری پائتھن کنونشنز کی پیروی کرتا ہو، جیسے کہ چھوٹے حروف اور انڈر سکورز کا استعمال کریں، مثلاً $simple_message.py$

اور $simple_messages.py$۔

- 2-1. سادہ پیغام: ایک پیغام کو کسی variable میں assign کریں، اور پھر اس پیغام کو پرنٹ کریں۔
- 2-2. سادہ پیغامات: ایک پیغام کو کسی variable میں assign کریں، اور اس پیغام کو پرنٹ کریں۔ پھر variable کی value کو ایک نئے پیغام میں تبدیل کریں، اور نئے پیغام کو پرنٹ کریں۔

Dr. Muhammad Siddique 03006329919

Strings

کیونکہ زیادہ تر پروگرامز کسی قسم کے ڈیٹا کو اکٹھا کرتے ہیں اور پھر اسے کسی مفید کام میں استعمال کرتے ہیں، ڈیٹا کی مختلف اقسام کو درجہ بندی کرنا مددگار ثابت ہوتا ہے۔ پہلی ڈیٹا قسم جس پر ہم غور کریں گے وہ string ہے۔ strings بظاہر سادہ نظر آتی ہیں لیکن انہیں کئی مختلف طریقوں سے استعمال کیا جا سکتا ہے۔

string کرداروں کی ایک ترتیب ہے۔ جو کچھ بھی کوٹس کے اندر ہو، اسے پائیتھون میں ایک string سمجھا جاتا ہے، اور آپ اپنی strings کے ارد گرد سنگل یا ڈبل کوٹس استعمال کر سکتے ہیں جیسے کہ:

```
"This is a string."
'This is also a string.'
```

یہ لچک آپ کو اپنی strings میں کوٹس اور اپوسٹروفی استعمال کرنے کی اجازت دیتی ہے:

```
'I told my friend, "Python is my favorite language!"'
"The language 'Python' is named after Monty Python, not the
    snake."
"One of Python's strengths is its diverse and supportive
    community."
```

Dr. Muhammad Siddique 03006329919

آیئے strings کو استعمال کرنے کے چند طریقے دیکھتے ہیں۔

Case کو Methods کے ساتھ تبدیل کرنا

سب سے آسان کاموں میں سے ایک، جو آپ strings کے ساتھ کر سکتے ہیں، ان کے -word- کا کیس تبدیل کرنا ہے۔ درج ذیل code دیکھیں اور سمجھنے کی کوشش کریں کہ کیا ہو رہا ہے:

```
name.py
name = "ada lovelace"
print(name.title())
```

اس فائل کو name.py کے طور پر محفوظ کریں اور اسے چلائیں۔ آپ کو یہ output دیکھنی چاہیے:

```
Ada Lovelace
```

اس مثال میں، name-variable لوئر کیس "adastring lovelace" کو ظاہر کرتا ہے۔ title()- method پرنٹ کال میں variable کے بعد ظاہر ہوتا ہے۔ method پائیتھون کی ایک کارروائی ہے جو ڈیٹا کے ایک ٹکڑے پر انجام دی جا سکتی ہے۔ . کے بعد name.title() پائیتھون کو بتاتا ہے کہ method-title() namevariable پر عمل کرے۔ ہر method کے بعد قوسین آتی ہیں کیونکہ اکثر method کو اپنا کام کرنے کے لیے اضافی معلومات کی ضرورت ہوتی ہے۔

method-title() ہر -word- کو ٹائٹل کیس میں تبدیل کرتا ہے، جہاں ہر -word- بڑے حرف سے شروع ہوتا ہے۔ یہ مفید ہے کیونکہ آپ اکثر کسی نام کو ایک معلومات کے ٹکڑے کے طور پر لینا چاہیں گے۔

strings میں variable کا استعمال

بعض Conditions میں، آپ string میں variable کی value-استعمال کرنا چاہیں گے۔ مثال کے طور پر:

```
first_name = "ada"
last_name = "lovelace"
full_name = f"{first_name} {last_name}"
print(f"Hello, {full_name.title()}!")
```

اوپر کے code کا output یہ ہوگا:

```
Hello, Ada Lovelace!
```

strings میں WhiteSpace شامل کرنا

پروگرامنگ میں WhiteSpace غیر پرنٹ ایبل کریکٹرز، جیسے کہ اسپیسز اور ٹیبز، کو کہتے ہیں۔ آپ اپنی output کو منظم کرنے کے لیے t\ یا n\ کا استعمال کر سکتے ہیں۔ مثال:

```
>>> print("Languages:\nPython\nC\nJavaScript")
Languages:
Python
C
JavaScript
```

Dr. Muhammad Siddique 03006339919

WhiteSpace کو ہٹانا

اضافی اسپیس کو ہٹانے کے لیے پائیتھون مختلف method فراہم کرتا ہے جیسے کہ rstrip()، lstrip()، اور strip()۔ یہ methods users کی جانب سے فراہم کردہ ڈیٹا کو صاف کرنے میں مدد دیتے ہیں۔

```
>>> favorite_language = 'python '
>>> favorite_language.rstrip()
'python'
>>> favorite_language.strip()
'python'
```

Prefixes ختم کرنا

جب ہم سٹرنگز کے ساتھ کام کرتے ہیں، تو ایک عام کام prefixes کو ختم کرنا ہوتا ہے۔ ایک URL پر غور کریں جس میں عام https://prefix شامل ہو۔ ہم اس prefix کو ہٹانا چاہتے ہیں تاکہ ہم صرف اس حصے پر توجہ مرکوز کر سکیں جو users ایڈریس بار میں داخل کرتے ہیں۔ اس کام کو کرنے کا طریقہ یہ ہے:

```
>>> nostarch_url = 'https://nostarch.com'
>>> nostarch_url.removeprefix('https://')
'nostarch.com'
```

variable کا نام لکھیں، اس کے بعد ایک نقطہ (dot) اور پھر method removeprefix()۔ گول بریکٹس کے اندر وہ prefix لکھیں جسے آپ اصل سٹرنگ سے ہٹانا چاہتے ہیں۔

جیسے whitespace ختم کرنے والے methods، removeprefix() بھی اصل سٹرنگ کو بغیر تبدیل کیے چھوڑ دیتا ہے۔ اگر آپ نئی value- کو محفوظ کرنا چاہتے ہیں، تو یا تو اسے اصل variable میں دوبارہ assign کریں یا کسی نئے variable میں محفوظ کریں:

```
>>> simple_url = nostarch_url.removeprefix('https://')
```

جب آپ ایڈریس بار میں ایک URL دیکھتے ہیں اور https:// حصہ ظاہر نہیں ہوتا، تو براؤزر شاید پردے کے پیچھے removeprefix() جیسا کوئی طریقہ استعمال کر رہا ہوتا ہے۔

Syntax-Errors کو روکنا

Dr. Muhammad Siddique 03006329919

ایک قسم کی غلطی جسے آپ اکثر دیکھ سکتے ہیں، وہ Syntax-error ہے۔ Syntax-error اس وقت ہوتی ہے جب Python آپ کے پروگرام کے کسی حصے کو درست Python-code کے طور پر نہیں پہچانتا۔ مثال کے طور پر، اگر آپ ایک apostrophe کو single-quotes کے اندر استعمال کریں گے، تو آپ کو ایک غلطی کا سامنا ہوگا۔ یہ اس لیے ہوتا ہے کیونکہ Python پہلے single-quote اور apostrophe کے درمیان ہر چیز کو ایک سٹرنگ کے طور پر تشریح کرتا ہے، اور پھر باقی متن کو Python-code کے طور پر پڑھنے کی کوشش کرتا ہے، جس کی وجہ سے غلطیاں ہوتی ہیں۔

درست طریقہ یہ ہے کہ single اور double quotes کو اس طرح استعمال کریں:

```
message = "One of Python's strengths is its diverse community."
print(message)
```

یہاں apostrophe-double-quotes کے اندر ظاہر ہوتی ہے، لہذا Python interpreter کو سٹرنگ کو صحیح طریقے سے پڑھنے میں کوئی مسئلہ نہیں ہوتا:

```
One of Python's strengths is its diverse community.
```

تاہم، اگر آپ single quotes استعمال کریں گے، تو Python اس بات کی نشاندہی نہیں کر سکے گا کہ سٹرنگ کہاں ختم ہونی چاہیے:

```
message = 'One of Python's strengths is its diverse community.'
print(message)
```

آپ کو درج ذیل output ملے گا:

```
File "apostrophe.py", line 1
message = 'One of Python's strengths is its diverse community.'
1 ^
SyntaxError: unterminated string literal (detected at line 1)
```

خود آزمائیں

مندرجہ ذیل وں excercise کو الگ الگ فائلوں میں محفوظ کریں اور اپنا code لکھیں:

1. ذاتی پیغام: ایک variable میں کسی person کا نام محفوظ کریں اور اس کے لیے ایک سادہ پیغام پرنٹ کریں۔
2. نام کی شکلیں: ایک variable میں کسی کا نام محفوظ کریں اور اسے، lowercase,-uppercase اور title-case میں پرنٹ کریں۔
3. مشہور اقتباس: کسی مشہور ریت person کا اقتباس تلاش کریں اور اسے پرنٹ کریں۔
4. فائل ایکسٹینشنز: method-removesuffix() استعمال کرتے ہوئے فائل کا نام بغیر ایکسٹینشن کے پرنٹ کریں۔

Dr. Muhammad Siddique 03006129919

اعداد و شمار(Numbers)

اعداد و شمار(Numbers)اکثر پروگرامنگ میں کھیلوں میں اسکور رکھنے، ڈیٹا کو بصری شکل میں ظاہر کرنے، ویب ایپلیکیشنز میں معلومات store کرنے، اور اسی طرح کے مقاصد کے لیے استعمال کیے جاتے ہیں۔ پائتھن اعداد کو مختلف طریقوں سے برتاؤ کرتا ہے، اس پر منحصر ہے کہ ان کا استعمال کیسے ہو رہا ہے۔ پہلے ہم دیکھتے ہیں کہ پائتھن کس طرح صحیح عدد(Integers) کو منظم کرتا ہے، کیونکہ ان کے ساتھ کام کرنا سب سے آسان ہے۔

صحیح عدد(Integers)

پائتھن میں آپ جمع(+)، تفریق(-)، ضرب(*)اور تقسیم(/) کے آپریشنز صحیح عدد کے ساتھ کر سکتے ہیں:

```
>>> 2 + 3
5
>>> 3 - 2
1
>>> 2 * 3
6
>>> 3 / 2
1.5
```

Dr. Muhammad Siddique 03006329919

ایک ٹرمینل سیشن میں، پائتھن آپریشن کا نتیجہ سادہ طور پر واپس کرتا ہے۔ پائتھن دو ضرب کی علامتوں کا استعمال طاقت(Exponents) ظاہر کرنے کے لیے کرتا ہے:

```
>>> 3 ** 2
9
>>> 3 ** 3
27
>>> 10 ** 6
1000000
```

پائتھن آپریشنز کی ترتیب Order)of(Operations کو بھی سپورٹ کرتا ہے، اس لیے آپ ایک ہی اظہار(Expression) میں متعدد آپریشنز استعمال کر سکتے ہیں۔ آپ ترتیب کو تبدیل کرنے کے لیے قوسین(Parentheses) کا استعمال بھی کر سکتے ہیں:

```
>>> 2 + 3 * 4
14
>>> (2 + 3) * 4
20
```

اعشاری اعداد(Floats)

پائتھن کسی بھی عدد کو جس میں اعشاریہ Decimal)(Point موجود ہو، اعشاری عدد(Float) کہتا ہے۔ یہ اصطلاح اکثر پروگرامنگ زبانوں میں استعمال ہوتی ہے۔ زیادہ تر معاملات میں، آپ اعشاری اعداد کو بغیر کسی پریشانی کے استعمال کر

سکتے ہیں:

```
>>> 0.1 + 0.1
0.2
>>> 0.2 + 0.2
0.4
>>> 2 * 0.1
0.2
>>> 2 * 0.2
0.4
```

لیکن کبھی کبھار آپ کے جواب میں اضافی اعشاری جگہیں ظاہر ہو سکتی ہیں:

```
>>> 0.2 + 0.1
0.30000000000000004
>>> 3 * 0.1
0.30000000000000004
```

صحیح عدد اور اعشاری عدد کا امتزاج

جب آپ دو اعداد کو تقسیم کرتے ہیں، چاہے وہ صحیح عدد ہوں اور نتیجہ مکمل عدد ہو، آپ کو ہمیشہ اعشاری عدد (Float) ملے گا:

Dr. Muhammad Siddique 03006329319

```
>>> 4 / 2
2.0
```

طویل اعداد میں انڈر سکور کا استعمال

طویل اعداد لکھتے وقت، آپ بڑے اعداد کو زیادہ قابل پڑھائی بنانے کے لیے انڈر سکور استعمال کر سکتے ہیں:

```
>>> universe_age = 14_000_000_000
>>> print(universe_age)
14000000000
```

متعدد variables کو ایک ساتھ مقدار دینا

آپ ایک ہی لائن میں ایک سے زیادہ variables کو مقدار دے سکتے ہیں:

```
>>> x, y, z = 0, 0, 0
```

مستقل variables (Constants)

ایک مستقل variable وہ ہے جس کی value- پروگرام کی زندگی کے دوران تبدیل نہیں ہوتی:

```
MAX_CONNECTIONS = 5000
```

Excercise

1. عدد آٹھ: ایسے حسابی آپریشن لکھیں جو نتیجے میں عدد آٹھ دیں۔ ہر آپریشن کو ()print میں لکھیں۔ مثلاً:

```
print(5 + 3)
```

1. پسندیدہ عدد: ایک variable میں اپنے پسندیدہ عدد کو store کریں، اور اس عدد کو ظاہر کرنے والا پیغام پرنٹ کریں۔

comments

comments زیادہ تر پروگرامنگ زبانوں میں ایک انتہائی مفید خصوصیت ہیں۔ اب تک آپ نے جو کچھ بھی اپنے پروگراموں میں لکھا ہے، وہ سب پائتھن code تھا۔

جب آپ کے پروگرام طویل اور زیادہ پیچیدہ ہو جائیں، تو آپ کو اپنے پروگراموں میں نوٹس شامل کرنے چاہییں جو آپ کے مسئلے کو حل کرنے کے مجموعی طریقے کی وضاحت کریں۔ ایک comment آپ کو اپنی مادری زبان میں پروگرام کے اندر نوٹس لکھنے کی اجازت دیتا ہے۔

Dr. Muhammad Siddique 03006329019

comments کیسے لکھے جاتے ہیں؟

پائتھن میں ہیش مارک (ایک comments کی نشاندہی کرتا ہے۔ آپ کے code میں ہیش مارک کے بعد کی ہر چیز کو پائتھن انٹرپریٹر نظر انداز کر دیتا ہے۔ مثال کے طور پر:

```
print("Hello Python people!")
```

```
Hello Python people!
```

کس قسم کے comments لکھنے چاہییں؟

comments لکھنے کی بنیادی وجہ یہ ہے کہ آپ کے code کا مقصد اور اس کے کام کرنے کا طریقہ واضح کیا جائے۔ جب آپ کسی منصوبے پر کام کر رہے ہوں، تو آپ کو معلوم ہوتا ہے کہ تمام حصے کیسے جڑتے ہیں۔ لیکن کچھ عرصے بعد منصوبے کی طرف لوٹنے پر آپ کچھ تفصیلات بھول سکتے ہیں۔

اچھے comments لکھنے سے وقت بچتا ہے کیونکہ یہ آپ کے مجموعی نقطہ نظر کو واضح طور پر statement-کرتے ہیں۔ اگر آپ ایک پیشہ ور پروگرامر بننا چاہتے ہیں یا دیگر پروگرامرز کے ساتھ تعاون کرنا چاہتے ہیں، تو آپ کو بامعنی comments لکھنے چاہییں۔

معقولیت پر مبنی comments لکھیں

جب آپ فیصلہ کر رہے ہوں کہ comment لکھنا ہے یا نہیں، تو خود سے پوچھیں کہ کیا آپ کو کچھ حل تلاش کرنے کے لیے مختلف طریقے آزمانے پڑے؟ اگر ہاں، تو اپنے حل کے بارے میں comment لکھیں۔ اضافی comments بعد میں حذف کرنا آسان ہے بہ نسبت ایسے پروگرام میں comments شامل کرنے کے جو بہت کم comments رکھتے ہوں۔

Excercise

1. دو پروگرام منتخب کریں جو آپ نے لکھے ہیں، اور ہر ایک میں کم از کم ایک comment شامل کریں۔ اگر آپ کے پروگرام بہت سادہ ہیں اور کچھ خاص لکھنے کے لیے نہیں ہے، تو ہر پروگرام کی فائل کے اوپر اپنا نام اور موجودہ تاریخ شامل کریں۔ پھر ایک جملے میں لکھیں کہ پروگرام کیا کرتا ہے۔

guiding-principles

ماہر پائتھن پروگرامرز تجویز کرتے ہیں کہ پیچیدگی سے بچیں اور جہاں ممکن ہو، سادگی کا انتخاب کریں۔ پائتھن کمیونٹی کی فلسفہ میں مضمر ہے جسے ٹم پیٹرز نے لکھا ہے۔ آپ اس فلسفے کو import this کمانڈ کے ذریعے دیکھ سکتے ہیں:

```
>>> import this
```

Some principles of The Zen of Python are:

- Beauty is better than ugly.
- Simplicity is better than complexity.
- Complexity is better than complicated.
- Readability counts.

Dr. Muhammad Siddique 03006329919

خلاصہ

اس باب میں، آپ نے variables کے ساتھ کام کرنا سیکھا اور comments لکھنے کی اہمیت کو سمجھا۔ اگلے باب میں آپ Lists کے بارے میں سیکھیں گے اور ان کے ذریعے معلومات store کرنے کے مختلف طریقے جانیں گے۔

Lists

اس باب اور اگلے باب میں آپ سیکھیں گے کہ یہ List کیا ہیں اور ان میں موجود elements کے ساتھ کام کرنا کیسے شروع کیا جائے۔ یہ List آپ کو معلومات کے سیٹ کو ایک جگہ پر store کرنے کی اجازت دیتی ہیں، چاہے آپ کے پاس صرف چند elements ہوں یا لاکھوں۔ یہ List پائتھن کی سب سے طاقتور خصوصیات میں سے ایک ہیں جو نئے پروگرامرز کے لئے دستیاب ہیں، اور یہ پروگرامنگ کے کئی اہم تصورات کو آپس میں جوڑتی ہیں۔

کیا ہے؟(List)

List ایک خاص ترتیب میں elements کا مجموعہ ہے۔ آپ ایک List بنا سکتے ہیں جس میں حروف تہجی کے حرف، 0 سے 9 تک کے عدد، یا آپ کے خاندان کے تمام افراد کے نام شامل ہوں۔ آپ List میں جو چاہیں ڈال سکتے ہیں، اور List میں شامل elements کو کسی خاص طریقے سے آپس میں جڑا ہونا ضروری نہیں۔ چونکہ List میں عموماً ایک سے زیادہ elements ہوتے ہیں، اس لئے یہ اچھا خیال ہے کہ آپ اپنی List کا نام جمع میں رکھیں جیسے حروف، اعداد، یا نام۔

پائتھن میں List کی نشاندہی square بریکٹس ([]) سے کی جاتی ہے، اور List کے انفرادی elements کو کاما سے الگ کیا جاتا ہے۔ یہاں ایک سادہ مثال ہے جس میں کچھ قسم کی bikes کی List شامل ہے:

Dr. Muhammad Siddique 03006329919

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)
```

اگر آپ پائتھن سے List کو پرنٹ کرنے کو کہیں گے تو پائتھن اس کی نمائندگی واپس کرتا ہے، جس میں square بریکٹس شامل ہیں:

```
['trek', 'cannondale', 'redline', 'specialized']
```

چونکہ یہ وہ output نہیں ہے جو آپ چاہتے ہیں کہ آپ کے users دیکھیں، تو آیئے سیکھتے ہیں کہ List کے انفرادی elements تک کیسے access حاصل کی جائے۔

List میں elements کو access حاصل

List ترتیب شدہ مجموعے ہوتی ہیں، اس لئے آپ کسی بھی element تک اس کی پوزیشن یا انڈیکس بتا کر access حاصل کر سکتے ہیں۔ List میں کسی element تک access حاصل کرنے کے لئے List کا نام لکھیں اور اس کے بعد انڈیکس کو square بریکٹس میں لکھیں۔

مثال کے طور پر، آیئے bicycles-List میں سے پہلی بائیک نکالتے ہیں:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[0])
```

جب ہم List سے ایک واحد element مانگتے ہیں، تو پائتھن صرف اسی element کو square بریکٹس کے بغیر واپس کرتا ہے:

```
trek
```

یہ وہ نتیجہ ہے جو آپ چاہتے ہیں کہ آپ کے users دیکھیں: صاف اور خوبصورتی سے فارمیٹ کیا ہوا output۔ آپ List کے کسی بھی element پر باب 2 کے string method بھی استعمال کر سکتے ہیں۔ مثال کے طور پر، آپ 'trek' element کو زیادہ پیشکش کے لئے method-title () استعمال کر کے فارمیٹ کر سکتے ہیں:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[0].-title())
```

یہ مثال پچھلی مثال کی طرح output پیدا کرتی ہے، سوائے اس کے کہ Trek کو کیپٹلائز کیا گیا ہے۔

انڈیکس کی پوزیشن 0 سے شروع ہوتی ہے، 1 سے نہیں

پائتھن میں List کا پہلا element پوزیشن 0 پر ہوتا ہے، نہ کہ پوزیشن 1 پر۔ یہ زیادہ تر پروگرامنگ زبانوں میں سچ ہے، اور اس کی وجہ یہ ہے کہ List کے آپریشنز کو کم سطح پر کس طرح عمل میں لایا جاتا ہے۔ اگر آپ کو غیر متوقع نتائج مل رہے ہیں، تو خود سے پوچھیں کہ کیا آپ ایک سادہ مگر عام غلطی کر رہے ہیں؟

List کا دوسرا element انڈیکس 1 پر ہوتا ہے۔ اس شمار کے نظام کا استعمال کرتے ہوئے، آپ List سے کوئی بھی element حاصل کر سکتے ہیں بس اس کی پوزیشن سے ایک کم کر کے۔

مثال کے طور پر، List میں چوتھے element کو access حاصل کرنے کے لئے آپ انڈیکس 3 کے مطابق element مانگیں گے۔

ذیل میں انڈیکس 1 اور 3 پر موجود بائیکوں کے لئے سوال کیا گیا ہے:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[1])
print(bicycles[3])
```

یہ code List میں دوسرے اور چوتھے بائیکوں کو واپس کرتا ہے:

```
cannondale
specialized
```

پائتھن میں List کے آخری element کو access حاصل کرنے کے لئے ایک خاص syntax موجود ہے۔ اگر آپ انڈیکس -1 سے element مانگتے ہیں تو پائتھن ہمیشہ List کا آخری element واپس کرتا ہے:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[-1])
```

یہ code 'specialized'-value واپس کرتا ہے۔ یہ syntax بہت مفید ہے، کیونکہ آپ اکثر List کے آخری elements تک access حاصل کرنا چاہتے ہیں بغیر یہ جانے کہ List کتنی لمبی ہے۔ یہ روایت دوسرے منفی

Dr. Muhammad Siddique 03006329919

انڈیکس کی-value وں پر بھی لاگو ہوتی ہے۔انڈیکس-2,List کے آخر سے دوسرا element واپس کرتا ہے،انڈیکس -3 تیسرے element کو واپس کرتا ہے،اور اسی طرح۔

List سے انفرادی-value یں استعمال کرنا

آپ List سے انفرادی-value وں کو اسی طرح استعمال کر سکتے ہیں جیسے آپ کسی دوسرے variable کو استعمال کرتے ہیں۔مثال کے طور پر،آپ List سے ایک-value استعمال کر کے ایک پیغام بنا سکتے ہیں۔
آیئے List میں سے پہلی بائیک نکال کر اس-value کو استعمال کرتے ہوئے ایک پیغام بناتے ہیں:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
message = f"My first bicycle was a {bicycles[0].-title()}."
print(message)
```

ہم List میں موجود value-bicycles[0] کو استعمال کر کے ایک جملہ بناتے ہیں اور اسے variable message میں محفوظ کرتے ہیں۔output ایک سادہ جملہ ہو گا:

```
My first bicycle was a Trek.
```

Dr. Muhammad Siddique 03006329919

Excercise

یہ مختصر پروگرام آزما کر پائتھن کی وں List کے ساتھ کچھ تجربہ حاصل کریں۔آپ ہر باب کی وں excercise کے لئے ایک نیا فولڈر بنانا چاہیں گے تاکہ انہیں منظم رکھا جا سکے۔

1. نام:اپنے کچھ دوستوں کے نام List میں store کریں جس کا نام names ہو۔List کے ہر element کو ایک بار access حاصل کر کے ہر person کا نام پرنٹ کریں۔

2. سلام:excercise 1-3 میں استعمال ہونے والی List سے شروع کریں،لیکن صرف ہر person کا نام پرنٹ کرنے کی بجائے،ان کے لئے ایک پیغام پرنٹ کریں۔ہر پیغام کا متن ایک ہی ہونا چاہئے،مگر ہر پیغام کو person کے نام سے ذاتی بنایا جائے۔

3. اپنی List بنائیں:اپنے پسندیدہ نقل و حمل کے طریقہ کا سوچیں،جیسے موٹر سائیکل یا گاڑی،اور ایک List بنائیں جس میں کئی مثالیں store کی گئی ہوں۔اس List کا استعمال کرتے ہوئے ان elements کے بارے میں ایک سیریز کے ات-statement پرنٹ کریں،جیسے ”میں ہونڈا موٹر سائیکل خریدنا چاہوں گا۔“

List Elements Delete Add, Modify,

جو زیادہ تر Lists آپ بنائیں گے وہ dynamic ہوں گی، یعنی آپ ایک List بنائیں گے اور پھر اپنے پروگرام کے چلتے رہنے کے دوران اس میں elements کو شامل اور ہٹاتے رہیں گے۔ مثال کے طور پر، آپ ایک کھیل بنا سکتے ہیں جس میں ایک کھلاڑی کو آسمان سے aliens کو گولی مارنی ہے۔ آپ aliens کا ابتدائی سیٹ ایک List میں محفوظ کر سکتے ہیں اور پھر جب ہر ایک alien کو گولی مار دی جائے تو اس List سے ایک alien کو ہٹا سکتے ہیں۔ جب بھی ایک نیا alien اسکرین پر آتا ہے، آپ اسے List میں شامل کر لیتے ہیں۔ آپ کی aliens کی List کھیل کے دوران لمبائی میں کم اور زیادہ ہوتی رہے گی۔

List میں elements کو ترمیم کرنا

کسی element کو ترمیم کرنے کا syntax (syntax) اس syntax کی طرح ہوتا ہے جو List میں کسی element access حاصل کو کے لیے استعمال کیا جاتا ہے۔ ایک element کو تبدیل کرنے کے لیے، List کا نام استعمال کریں، پھر اس element کا انڈیکس دیں جسے آپ تبدیل کرنا چاہتے ہیں، اور پھر اس نئے value کو فراہم کریں جو آپ چاہتے ہیں کہ وہ شے حاصل کرے۔

مثال کے طور پر، فرض کریں ہمارے پاس موٹر سائیکلوں کی ایک List ہے اور List کا پہلا element 'ہونڈا' ہے۔ ہم List کے بننے کے بعد اس پہلے element کی value-کو تبدیل کر سکتے ہیں:

Dr. Muhammad Siddique 03006329919

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
motorcycles[0] = 'ducati'
print(motorcycles)
```

یہاں ہم List 'موٹر سائیکلوں' کی declare کرتے ہیں، جس میں 'ہونڈا' پہلا element ہوتا ہے۔ پھر ہم پہلے element کی value-'ڈوکٹی' میں تبدیل کر دیتے ہیں۔ output یہ دکھاتا ہے کہ پہلا element تبدیل ہو چکا ہے، جبکہ باقی List میں کوئی تبدیلی نہیں آئی:

```
['honda', 'yamaha', 'suzuki']
['ducati', 'yamaha', 'suzuki']
```

آپ List میں کسی بھی شے کی value- تبدیل کر سکتے ہیں، نہ کہ صرف پہلے element کی۔

List میں elements شامل کرنا

آپ کو کئی وجوہات کی بنا پر List میں نیا element شامل کرنے کی ضرورت ہو سکتی ہے۔ مثال کے طور پر، آپ چاہتے ہیں کہ ایک کھیل میں نئے alien ظاہر ہوں، ڈیٹا کی ایک نیا سیٹ کسی بصری نمائندگی میں شامل کریں، یا کسی ویب سائٹ پر نئے رجسٹرڈ users کو شامل کریں۔ Python نئے ڈیٹا کو موجودہ وں List میں شامل کرنے کے لیے کئی طریقے فراہم کرتا ہے۔

List کے آخر میں elements شامل کرنا

List میں نیا element شامل کرنے کا سب سے آسان طریقہ ہے کہ آپ اس element کو List کے آخر میں شامل کریں۔ جب آپ کسی element کو List میں شامل کرتے ہیں، تو وہ نیا element List کے آخر میں آجاتا ہے۔ پچھلے مثال کے List کو استعمال کرتے ہوئے، ہم نیا element 'ڈوکٹی' List کے آخر میں شامل کرتے ہیں:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
motorcycles.append('ducati')
print(motorcycles)
```

یہاں append() طریقہ 'ڈوکٹی' کو List کے آخر میں شامل کرتا ہے، بغیر باقی elements کو متاثر کیے:

```
['honda', 'yamaha', 'suzuki']
['honda', 'yamaha', 'suzuki', 'ducati']
```

append() طریقہ وں List کو dynamic طور پر بنانے میں آسانی فراہم کرتا ہے۔ مثال کے طور پر، آپ ایک ListEmpty سے شروع کر سکتے ہیں اور پھر append() طریقے کا استعمال کرتے ہوئے List میں elements شامل کر سکتے ہیں۔

Dr. Muhammad Siddique 03006329919

List میں elements داخل کرنا

آپ insert() طریقے کا استعمال کرتے ہوئے List میں کسی بھی مقام پر نیا element شامل کر سکتے ہیں۔ آپ یہ اس طرح کرتے ہیں کہ نئے element کا انڈیکس اور اس کا نیا-value فراہم کرتے ہیں:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
motorcycles.insert(0, 'ducati')
print(motorcycles)
```

اس مثال میں، ہم 'ڈوکٹی' کی-value List کے آغاز میں داخل کرتے ہیں۔ insert() طریقہ 0 مقام پر جگہ کھولتا ہے اور اس جگہ پر 'ڈوکٹی' کی-value store کرتا ہے:

```
['ducati', 'honda', 'yamaha', 'suzuki']
```

یہ آپریشن List میں ہر دوسری-value کو ایک پوزیشن دائیں طرف منتقل کر دیتا ہے۔

List سے elements کو ہٹانا

اکثر آپ کو List سے کسی شے یا کچھ elements کو ہٹانے کی ضرورت ہو گی۔ مثال کے طور پر، جب کھلاڑی آسمان سے ایک alien کو گولی مار دیتا ہے، تو آپ عموماً اس alien کو List سے ہٹانا چاہیں گے۔ یا جب کوئی User اپنی ویب ایپلیکیشن پر اپنا اکاؤنٹ منسوخ کرتا ہے، تو آپ چاہیں گے کہ اس User کو List سے ہٹا دیا جائے۔

statement-del کا استعمال کرتے ہوئے ایک شے کو ہٹانا

اگر آپ جانتے ہیں کہ جس شے کو آپ List سے ہٹانا چاہتے ہیں اس کا مقام کہاں ہے، تو آپ statement-del کا استعمال کر سکتے ہیں:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
del motorcycles[0]
print(motorcycles)
```

یہاں ہم statement-del کا استعمال کرتے ہوئے List میں سے پہلا element 'ہونڈا' ہٹا دیتے ہیں:

```
['honda', 'yamaha', 'suzuki']
['yamaha', 'suzuki']
```

آپ List میں کسی بھی مقام سے ایک شے کو statement-del کے ذریعے ہٹا سکتے ہیں اگر آپ اس کے انڈیکس کو جانتے ہوں۔

Dr. Muhammad Siddique 03006329919

List کو منظم کرنا

اکثر، آپ کی یں List ایک غیر متوقع ترتیب میں تخلیق کی جاتی ہیں، کیونکہ آپ ہمیشہ اس ترتیب کو قابو نہیں کر سکتے جس میں آپ کے users اپنا ڈیٹا فراہم کرتے ہیں۔ اگرچہ زیادہ تر Conditions میں یہ ناگزیر ہے، آپ کو اکثر اپنی معلومات کو ایک مخصوص ترتیب میں پیش کرنے کی ضرورت ہوگی۔ کبھی کبھار آپ چاہیں گے کہ آپ کی List کی اصل ترتیب برقرار رہے، اور کبھی کبھار آپ چاہیں گے کہ آپ اصل ترتیب کو تبدیل کریں۔ Python مختلف طریقوں سے آپ کی وں List کو ترتیب دینے کے لیے کئی طریقے فراہم کرتا ہے، جو کہ صورت حال پر منحصر ہیں۔

List کو مستقل طور پر sort()-method کے ساتھ ترتیب دینا

Python کا sort-method-() کو ترتیب دینا نسبتاً آسان بناتا ہے۔ تصور کریں کہ ہمارے پاس گاڑیوں کی ایک List ہے اور ہم چاہتے ہیں کہ اس List کو Alphabet-طور پر ترتیب دیں۔ اس کام کو سادہ رکھنے کے لیے، ہم فرض کرتے ہیں کہ List میں تمام values چھوٹے حروف میں ہیں:

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
cars.sort()
print(cars)
```

Dr. Muhammad Siddique 03006329919

sort-method-() کی ترتیب کو مستقل طور پر تبدیل کر دیتا ہے۔ گاڑیاں اب Alphabet-ترتیب میں ہیں، اور ہم کبھی بھی اصل ترتیب پر واپس نہیں جا سکتے:

```
['audi', 'bmw', 'subaru', 'toyota']
```

آپ اس List کو Alphabet-reverse-ترتیب میں بھی ترتیب دے سکتے ہیں، جس کے لیے sort-method-() کو reverse=True آرگومنٹ کے ساتھ استعمال کریں:

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
cars.sort(reverse=True)
print(cars)
```

ایک بار پھر، List کی ترتیب مستقل طور پر تبدیل ہو جاتی ہے:

```
['toyota', 'subaru', 'bmw', 'audi']
```

List کو عارضی طور پر sorted() فنکشن کے ساتھ ترتیب دینا

اگر آپ List کی اصل ترتیب کو برقرار رکھتے ہوئے اسے ترتیب دینا چاہتے ہیں تو آپ sorted-() فنکشن استعمال کر سکتے ہیں۔ sorted-() فنکشن آپ کو اپنی List کو ایک مخصوص ترتیب میں دکھانے کی اجازت دیتا ہے، لیکن یہ List کی اصل ترتیب پر اثر انداز نہیں ہوتا۔

آیئے ہم اس فنکشن کو گاڑیوں کی List پر آزمانے کی کوشش کریں:

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
print("Here is the original list:")
print(cars)
print("\nHere is the sorted list:")
print(sorted(cars))
print("\nHere is the original list again:")
print(cars)
```

ہم پہلے List کو اس کی اصل ترتیب میں پرنٹ کرتے ہیں، پھر Alphabet-ترتیب میں پرنٹ کرتے ہیں۔ List نئے ترتیب میں دکھانے کے بعد، ہم یہ ظاہر کرتے ہیں کہ List اب بھی اپنی اصل ترتیب میں ہے:

```
Here is the original list:
['bmw', 'audi', 'toyota', 'subaru']
Here is the sorted list:
['audi', 'bmw', 'subaru', 'toyota']
Here is the original list again:
['bmw', 'audi', 'toyota', 'subaru']
```

نوٹ کریں کہ List sorted() فنکشن کے استعمال کے بعد بھی اپنی اصل ترتیب میں موجود ہے۔ sorted() فنکشن کو reverse=True آرگومنٹ بھی دیا جا سکتا ہے اگر آپ List کو Alphabet-reverse- ترتیب میں دکھانا چاہتے ہیں۔

Dr. Muhammad Siddique 03006339919

List کو reverse- ترتیب میں پرنٹ کرنا

اگر آپ List کی اصل ترتیب کو الٹنا چاہتے ہیں تو آپ reverse() methodاستعمال کر سکتے ہیں۔ اگر ہم نے ابتدا میں گاڑیوں کی List کو کرونولوجیکل ترتیب میں store کیا تھا تو ہم آسانی سے List کو reverse- کرونولوجیکل ترتیب میں ترتیب دے سکتے ہیں:

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
print(cars)
cars.reverse()
print(cars)
```

نوٹ کریں کہ reverse() Alphabet- طور پر پیچھے ترتیب نہیں دیتا؛ یہ صرف List کی ترتیب کو الٹ دیتا ہے:

```
['bmw', 'audi', 'toyota', 'subaru']
['subaru', 'toyota', 'audi', 'bmw']
```

reverse() method- کی ترتیب کو مستقل طور پر تبدیل کر دیتا ہے، لیکن آپ جب چاہیں اسی List پر reverse() کو دوبارہ استعمال کر کے اصل ترتیب پر واپس جا سکتے ہیں۔

List کی لمبائی معلوم کرنا

آپ List کی لمبائی len() فنکشن کا استعمال کر کے جلدی معلوم کر سکتے ہیں۔ اس مثال میں List میں چار elements ہیں، اس لیے اس کی لمبائی 4 ہے:

```
>>> cars = ['bmw', 'audi', 'toyota', 'subaru']
>>> len(cars)
4
```

len() آپ کو اس وقت مفید ہو گا جب آپ کو کسی کھیل میں ابھی تک گرائے جانے والی اجسام کی تعداد معلوم کرنی ہو، یا کسی ویژولائزیشن میں آپ کو ڈیٹا کی مقدار جاننی ہو، یا ویب سائٹ پر رجسٹرڈ users کی تعداد معلوم کرنی ہو۔

Excercise

دنیا کا مشاہدہ: دنیا میں کم از کم پانچ جگہوں کے بارے میں سوچیں جہاں آپ جانا چاہتے ہیں۔

- مقامات کو ایک List میں محفوظ کریں۔ یہ یقینی بنائیں کہ List حروف تہجی کے مطابق نہ ہو۔
- اپنی List کو اس کی اصل ترتیب میں پرنٹ کریں۔ List کو صاف ستھرا پرنٹ کرنے کی فکر نہ کریں؛ بس اسے خام پائیتھن کی List کے طور پر پرنٹ کریں۔
- sorted() کا استعمال کریں تاکہ آپ کی List کو حروف تہجی کے مطابق پرنٹ کیا جا سکے بغیر اصل List کو تبدیل کیے۔
- اپنی List کو دوبارہ پرنٹ کر کے یہ دکھائیں کہ یہ اب بھی اس کی اصل ترتیب میں ہے۔
- sorted() کا استعمال کریں تاکہ آپ کی List کو الٹ حروف تہجی کے مطابق پرنٹ کیا جا سکے بغیر اصل List کی ترتیب کو بدلے۔
- اپنی List کو دوبارہ پرنٹ کر کے یہ دکھائیں کہ یہ اب بھی اس کی اصل ترتیب میں ہے۔
- reverse() کا استعمال کریں تاکہ آپ کی List کی ترتیب کو تبدیل کیا جا سکے۔ List کو پرنٹ کریں تاکہ یہ دکھایا جا سکے کہ اس کی ترتیب بدل چکی ہے۔
- reverse() کا استعمال کر کے List کی ترتیب کو دوبارہ تبدیل کریں۔ List کو پرنٹ کریں تاکہ یہ دکھایا جا سکے کہ یہ دوبارہ اس کی اصل ترتیب میں آ گئی ہے۔
- sort() کا استعمال کریں تاکہ List کو حروف تہجی کے مطابق ترتیب دیا جا سکے۔ List کو پرنٹ کریں تاکہ یہ دکھایا جا سکے کہ اس کی ترتیب بدل چکی ہے۔

Dr. Muhammad Siddique 03006329918

- sort()کا استعمال کریں تاکہ Listکو الٹ حروف تہجی کے مطابق ترتیب دیا جاسکے۔Listکو پرنٹ کریں تاکہ یہ دکھایا جاسکے کہ اس کی ترتیب بدل چکی ہے۔

ہر فنکشن: سوچیں کہ آپ کس چیز کو Listمیں محفوظ کر سکتے ہیں۔مثال کے طور پر، آپ پہاڑوں، دریاؤں، ممالک، شہروں، زبانوں یا کوئی اور چیز جو آپ چاہیں، کی Listبنا سکتے ہیں۔ایک پروگرام لکھیں جو ان elementsکو محفوظ کرنے والی Listبنائے اور پھر اس باب میں متعارف کرائے گئے ہر فنکشن کو کم از کم ایک بار استعمال کرے۔

Listsکے ساتھ کام کرتے وقت انڈیکس کی غلطیوں سے بچنا

ایک قسم کی غلطی ہے جو Listsکے ساتھ کام کرتے وقت عام طور پر نظر آتی ہے جب آپ پہلی بار Listsکے ساتھ کام کر رہے ہوتے ہیں۔فرض کریں کہ آپ کے پاس تین elementsوالی ایک Listہے، اور آپ چوتھی چیز مانگتے ہیں:

```
motorcycles.py
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles[3])
```

اس مثال میں انڈیکس کی غلطی آتی ہے:

```
Traceback (most recent call last):
File "motorcycles.py", line 2, in <module>
print(motorcycles[3])
~~~~~~~~~~~^^^
IndexError: list index out of range
```

Dr. Muhammad Siddique 03006329919

پائیتھن انڈیکس 3پر موجود elementsکو دینے کی کوشش کرتا ہے، لیکن جب یہ Listکو تلاش کرتا ہے، تو 'motorcycles' میں کوئی بھی شے انڈیکس 3پر نہیں ہوتی۔Listsمیں انڈیکسنگ کی نوعیت کی وجہ سے یہ غلطی عام ہے۔لوگ سمجھتے ہیں کہ تیسری چیز آئٹم نمبر 3ہے کیونکہ وہ گنتی 1سے شروع کرتے ہیں، لیکن پائیتھن میں تیسری شے آئٹم نمبر 2ہے کیونکہ یہ انڈیکسنگ 0سے شروع کرتا ہے۔

ایک انڈیکس کی غلطی کا مطلب ہے کہ پائیتھن آپ کے درخواست کردہ انڈیکس پر کوئی شے نہیں پا رہا۔اگر آپ کے پروگرام میں انڈیکس کی غلطی آتی ہے، تو آپ جو انڈیکس مانگ رہے ہیں اسے ایک کے حساب سے ایڈجسٹ کرنے کی کوشش کریں، پھر پروگرام کو دوبارہ چلائیں تاکہ یہ دیکھ سکیں کہ آیا نتائج درست ہیں۔

یاد رکھیں کہ جب بھی آپ Listمیں آخری شے تک access حاصل کرنا چاہیں، آپ کو -1انڈیکس استعمال کرنا چاہیے۔یہ ہمیشہ کام کرے گا، چاہے آپ کی Listکا سائز آخری بار آپ کے اسے access حاصل کرنے کے بعد تبدیل ہو چکا ہو:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles[-1])
```

1-انڈیکس ہمیشہ List میں آخری شے کو واپس کرتا ہے، اس معاملے میں 'suzuki' کی value-:

```
suzuki
```

یہ طریقہ صرف اس صورت میں غلطی پیدا کرے گا جب آپ ایک ListEmpty سے آخری شے کو درخواست دیں:

```
motorcycles = []
print(motorcycles[-1])
```

'Motorcycles' میں کوئی بھی شے موجود نہیں ہے، لہذا پائیتھن ایک اور انڈیکس کی غلطی واپس کرتا ہے:

```
Traceback (most recent call last):
File "motorcycles.py", line 3, in <module>
print(motorcycles[-1])
~~~~~~~~~~~^^^^
IndexError: list index out of range
```

اگر انڈیکس کی غلطی آتی ہے اور آپ نہیں سمجھ پا رہے ہیں کہ اسے کیسے حل کیا جائے، تو اپنی List کو پرنٹ کرنے یا صرف اپنی List کی لمبائی پرنٹ کرنے کی کوشش کریں۔ آپ کی List وہ نہیں ہو سکتی جو آپ نے سوچا تھا، خاص طور پر اگر اسے آپ کے پروگرام کے ذریعے ڈائنامک طور پر منظم کیا گیا ہو۔ اصل List کو دیکھنا، یا List میں موجود elements کی صحیح تعداد دیکھنا آپ کو اس قسم کی منطقی غلطیوں کو درست کرنے میں مدد دے سکتا ہے۔

Dr. Muhammad Siddique 03006329919

Excercise

جان بوجھ کر غلطی: اگر آپ نے اپنے کسی پروگرام میں ابھی تک انڈیکس کی غلطی نہیں دیکھی، تو اسے پیدا کرنے کی کوشش کریں۔ اپنے پروگرام میں کسی انڈیکس کو تبدیل کریں تاکہ انڈیکس کی غلطی پیدا ہو۔ اس بات کو یقینی بنائیں کہ آپ پروگرام بند کرنے سے پہلے غلطی کو درست کر لیں۔

خلاصہ

اس باب میں، آپ نے سیکھا کہ یں List کیا ہیں اور انفرادی elements کے ساتھ کام کیسے کیا جاتا ہے۔ آپ نے سیکھا کہ List کو کیسے statement- کیا جائے اور elements کو کیسے شامل اور حذف کیا جائے۔ آپ نے سیکھا کہ Lists کو مستقل اور عارضی طور پر ترتیب دینے کا طریقہ کیا ہے تاکہ انہیں دکھایا جا سکے۔ آپ نے یہ بھی سیکھا کہ List کی لمبائی کیسے معلوم کی جائے اور Lists کے ساتھ کام کرتے وقت انڈیکس کی غلطیوں سے کیسے بچا جائے۔

اگلے باب میں آپ سیکھیں گے کہ List میں موجود elements کے ساتھ زیادہ مؤثر طریقے سے کیسے کام کیا جائے۔ صرف چند لائنوں کا code استعمال کرتے ہوئے List میں موجود ہر شے کے ساتھ Looping کر کے آپ مؤثر طریقے سے کام کر سکیں گے، یہاں تک کہ جب آپ کی List میں ہزاروں یا لاکھوں elements ہوں۔

Indentation کی غلطیوں سے بچنا

پائیتھون Indentation کو استعمال کرتا ہے تاکہ یہ پتہ چل سکے کہ ایک لائن یا لائنوں کا گروپ پروگرام کے باقی حصے سے کس طرح متعلق ہے۔ پچھلے مثالوں میں، وہ لائنیں جو ہر جادوگر کو پیغام پرنٹ کرتی تھیں وہ Loop for کا حصہ تھیں کیونکہ وہ انڈینٹڈ تھیں۔ پائیتھون کا Indentation کا استعمال code کو پڑھنے میں بہت آسان بناتا ہے۔ بنیادی طور پر، یہ سفید جگہوں کا استعمال کرتا ہے تاکہ آپ کو صاف ستھرا code لکھنے پر مجبور کرے جو واضح بصری ڈھانچہ رکھتا ہو۔ طویل پائیتھون پروگرامز میں آپ کو مختلف سطحوں پر انڈنٹڈ code کے بلاکس نظر آئیں گے۔ یہ Indentation کی سطحیں آپ کو پروگرام کی مجموعی تنظیم کا اندازہ دینے میں مدد دیتی ہیں۔

جب آپ ایسا code لکھنا شروع کرتے ہیں جو صحیح Indentation پر منحصر ہو، تو آپ کو چند عام Indentation کی غلطیوں کا سامنا ہو سکتا ہے۔ مثال کے طور پر، لوگ بعض اوقات ایسی لائنوں کو انڈینٹ کرتے ہیں جنہیں انڈینٹ کرنے کی ضرورت نہیں ہوتی یا ان لائنوں کو انڈینٹ کرنا بھول جاتے ہیں جنہیں انڈینٹ کرنے کی ضرورت ہوتی ہے۔ ان غلطیوں کے مثالیں دیکھنا آپ کو ان سے بچنے میں مدد دے گا اور جب یہ آپ کے پروگرامز میں ظاہر ہوں تو آپ انہیں درست کر سکیں گے۔

آیئے کچھ عام Indentation کی غلطیوں کا جائزہ لیتے ہیں۔

Dr. Muhammad Siddique 03065329919

Indentation کرنا بھولنا

ہمیشہ for اسٹیٹمنٹ کے بعد لائن کو انڈینٹ کریں۔ اگر آپ بھول جائیں تو پائیتھون آپ کو یاد دلاتا ہے:

```
magicians.py
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(magician)
```

پہلا print() کو انڈینٹ کیا جانا چاہئے تھا، مگر ایسا نہیں کیا گیا۔ جب پائیتھون کو انڈینٹڈ بلاک کی توقع ہوتی ہے اور وہ نہیں ملتا، تو یہ آپ کو بتاتا ہے کہ یہ کس لائن میں مسئلہ تھا:

```
File "magicians.py", line 3
print(magician)
^
IndentationError: expected an indented block after 'for'
    statement on line 2
```

آپ اس قسم کی Indentation کی غلطی کو عام طور پر فوراً اسٹیٹمنٹ کے بعد لائن یا لائنوں کو انڈینٹ کر کے حل کر سکتے ہیں۔

اضافی لائنوں کو انڈینٹ کرنا بھولنا

کبھی کبھار آپ کا Loop کسی غلطی کے بغیر چلتا رہتا ہے لیکن متوقع نتیجہ پیدا نہیں ہوتا۔ یہ اس وقت ہو سکتا ہے جب آپ Loop میں کئی کام کرنے کی کوشش کرتے ہیں اور آپ کچھ لائنوں کو انڈینٹ کرنا بھول جاتے ہیں۔
مثال کے طور پر، یہ اس وقت ہوتا ہے جب ہم دوسرے print() کال کو انڈینٹ کرنا بھول جاتے ہیں جو ہر جادوگر کو ان کے اگلے جادو کے لیے انتظار کر رہا ہوتا ہے:

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(f"{magician.-title()}, that was a great trick!")
print(f"I can't wait to see your next trick, {magician.-title()
    }.\n")
```

دوسری print() کال انڈینٹ ہونی چاہیے تھی، مگر کیونکہ پائیتھون فور اسٹیٹمنٹ کے بعد کم از کم ایک انڈینٹڈ لائن پاتا ہے، یہ غلطی نہیں بتاتا۔ اس کے نتیجے میں، پہلی print() کال ہر نام کے لیے ایک بار چلتی ہے کیونکہ وہ انڈینٹڈ ہے۔ دوسری print() کال انڈینٹڈ نہیں ہے، اس لیے یہ صرف ایک بار چلتی ہے جب Loop مکمل ہو چکا ہوتا ہے۔

Dr. Muhammad Siddique 03006329919

غیر ضروری Indentation

اگر آپ غلطی سے ایسی لائن کو انڈینٹ کر دیتے ہیں جسے انڈینٹ کرنے کی ضرورت نہیں ہے، تو پائیتھون آپ کو غیر متوقع Indentation کی اطلاع دیتا ہے:

```
hello_world.py
message = "Hello Python world!"
print(message)
```

ہمیں print() کو انڈینٹ کرنے کی ضرورت نہیں ہے کیونکہ یہ کسی Loop کا حصہ نہیں ہے؛ لہذا پائیتھون اس غلطی کی اطلاع دیتا ہے:

```
File "hello_world.py", line 2
print(message)
^
IndentationError: unexpected indent
```

غیر متوقع Indentation کی غلطیوں سے بچنے کے لیے آپ کو صرف اس صورت میں انڈینٹ کرنا چاہیے جب آپ کے پاس ایسا کرنے کی کوئی خاص وجہ ہو۔

Loop کے بعد غیر ضروری Indentation

اگر آپ غلطی سے ایسے code کو انڈینٹ کر دیتے ہیں جو Loop کے بعد چلنا چاہیے، تو یہ code ہر آئٹم کے لیے ایک بار دوبارہ چلے گا۔ کبھی کبھار یہ پائیتھون کو غلطی کی اطلاع دینے پر مجبور کرتا ہے، لیکن اکثر یہ ایک منطقی غلطی پیدا کرتا ہے۔

```
magicians.py
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(f"{magician.-title()}, that was a great trick!")
    print(f"I can't wait to see your next trick, {magician.-
   title()}.\n")
    print("Thank you everyone, that was a great magic show!")
```

چونکہ آخری لائن انڈینٹڈ ہے، یہ ہر person کے لیے ایک بار پرنٹ ہو گی۔

کولن بھولنا

for اسٹیٹمنٹ کے آخر میں کولن پائیتھون کو بتاتا ہے کہ اگلی لائن Loop کے آغاز کے طور پر سمجھی جائے۔

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians
    print(magician)
```

اگر آپ کولن بھول جاتے ہیں تو پائیتھون آپ کو ایک syntax کی غلطی دکھائے گا کیونکہ یہ نہیں جانتا کہ آپ کیا کرنے کی کوشش کر رہے ہیں:

```
File "magicians.py", line 2
for magician in magicians
^
SyntaxError: expected ':'
```

Dr. Muhammad Siddique 03006329919

Numeric-List بنانا

نمبرز کے سیٹ کو store کرنے کی کئی وجوہات ہیں۔ مثال کے طور پر، آپ کو کسی کھیل میں ہر کردار کی پوزیشنز کو ٹریک کرنا پڑے گا، اور آپ کھلاڑی کے ہائی اسکورز کو بھی ٹریک کرنا چاہیں گے۔ ڈیٹا کی ویژولائزیشنز میں، آپ کو تقریباً ہمیشہ نمبرز کے سیٹس کے ساتھ کام کرنا پڑے گا، جیسے درجہ حرارت، فاصلہ، آبادی کے حجم یا طول و عرض کی value-s، اور دیگر اقسام کے عددی سیٹس۔

Lists نمبرز کے سیٹس کو store کرنے کے لیے بہترین ہیں، اور Python آپ کو نمبرز کی Lists کے ساتھ مؤثر طریقے سے کام کرنے کے لیے مختلف ٹولز فراہم کرتا ہے۔ جب آپ ان ٹولز کو مؤثر طریقے سے استعمال کرنا سیکھ جائیں گے، تو آپ کا code بہترین کام کرے گا چاہے آپ کی List میں لاکھوں آئٹمز ہوں۔

range() فنکشن کا استعمال

Python کا range () فنکشن نمبرز کی سیریز پیدا کرنا بہت آسان بناتا ہے۔ مثال کے طور پر، آپ range () فنکشن کا استعمال کر کے نمبرز کی سیریز اس طرح پرنٹ کر سکتے ہیں:

```
for value in range(1, 5):
    print(value)
```

اگرچہ یہ code 1 سے 5 تک کے نمبر پرنٹ کرنے جیسا لگتا ہے، لیکن یہ نمبر 5 پرنٹ نہیں کرتا:

```
1
2
3
4
```

اس مثال میں، range () صرف 1 سے 4 تک کے نمبر پرنٹ کرتا ہے۔ یہ ایک اور نتیجہ ہے جو اکثر پروگرامنگ زبانوں میں off-by-one غلطی کی صورت میں آتا ہے۔ range () فنکشن Python کو یہ حکم دیتا ہے کہ وہ آپ کے دیے گئے پہلے نمبر سے گنتی شروع کرے اور دوسرے نمبر تک پہنچتے ہی رک جائے۔ چونکہ یہ دوسرے نمبر پر رک جاتا ہے، اس لیے output میں آخری نمبر شامل نہیں ہوتا، جو کہ اس مثال میں 5 ہوتا۔

1 سے 5 تک کے نمبر پرنٹ کرنے کے لیے آپ range (1,6) استعمال کریں گے:

```
for value in range(1, 6):
    print(value)
```

اس مرتبہ output-1 سے شروع ہوکر 5 پر ختم ہوتا ہے:

```
1
2
3
4
5
```

Dr. Muhammad Siddique 03006325919

آپ range () کو صرف ایک آرگومنٹ بھی دے سکتے ہیں، اور یہ نمبر کی سیریز 0 سے شروع کرے گا۔ مثال کے طور پر، range (6) نمبرز کو 0 سے 5 تک واپس کرے گا۔

range() کا استعمال نمبرز کی List بنانے کے لیے

اگر آپ نمبرز کی List بنانا چاہتے ہیں، تو آپ range () کے نتائج کو براہ راست list () فنکشن کے ذریعے List میں تبدیل کر سکتے ہیں۔ جب آپ list () کو range () فنکشن کے ارد گرد لپیٹتے ہیں، تو output ایک نمبرز کی List بن جائے گی۔

پچھلی مثال میں، ہم نے صرف نمبرز کی سیریز پرنٹ کی تھی۔ ہم list () کا استعمال کرتے ہوئے ان ہی نمبرز کو List میں تبدیل کر سکتے ہیں:

```
numbers = list(range(1, 6))
print(numbers)
```

یہ نتیجہ ہوگا:

```
[1, 2, 3, 4, 5]
```

ہم range () فنکشن کا استعمال کر کے Python کو کسی مخصوص رینج میں نمبرز کو چھوڑنے کا بھی کہہ سکتے ہیں۔ اگر آپ range () کو تیسرا آرگومنٹ دیتے ہیں، تو Python اس value- کو قدم کے سائز کے طور پر استعمال کرتا ہے جب وہ نمبرز پیدا کرتا ہے۔

مثال کے طور پر، یہاں 1 سے 10 تک کے ایون نمبرز کو List کرنے کا طریقہ ہے:

```
even_numbers = list(range(2, 11, 2))
print(even_numbers)
```

اس مثال میں، range () فنکشن value- 2 سے شروع ہوتا ہے اور پھر اس میں 2 کا اضافہ کرتا ہے۔ یہ 2 کو بار بار بڑھاتا ہے جب تک یہ اختتام تک نہیں پہنچتا، اور یہ نتیجہ پیدا کرتا ہے:

```
[2, 4, 6, 8, 10]
```

آپ range () فنکشن کا استعمال کرتے ہوئے تقریباً کوئی بھی نمبرز کا سیٹ بنا سکتے ہیں۔ مثال کے طور پر، فرض کریں کہ آپ پہلے 10 square نمبرز کی List بنانا چاہتے ہیں (یعنی 1 سے 10 تک کے ہر عدد کا۔ (square Python میں دو ایسٹیرک (**) نمایاں کرتے ہیں۔

یہاں آپ پہلے 10 square نمبرز کو List میں کیسے ڈال سکتے ہیں:

```
squares = []
for value in range(1, 11):
    square = value ** 2
    squares.append(square)
print(squares)
```

Dr. Muhammad Siddique 03006329967

ہم ایک Empty-squares-List سے شروع کرتے ہیں۔ پھر، ہم Python کو 1 سے 10 تک ہر نمبر پر Loop کرنے کا حکم دیتے ہیں۔ Loop کے اندر، موجودہ نمبر کو دوسرے طاقت میں اٹھایا جاتا ہے اور square variable کو assign کیا جاتا ہے۔ پھر ہر نیا square کی value- Listsquares میں شامل کی جاتی ہے۔ آخر کار، جب Loop مکمل ہو جاتا ہے، square نمبروں کی List پرنٹ کی جاتی ہے:

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

اس code کو زیادہ مختصر لکھنے کے لیے، عارضی variable- square کو حذف کر کے ہر نئی value- کو سیدھا List میں شامل کر سکتے ہیں:

```
squares = []
for value in range(1, 11):
    squares.append(value ** 2)
print(squares)
```

یہ لائن پچھلی List میں Loop کے اندر موجود لائنوں کی طرح ہی کام کرتی ہے۔ Loop میں ہر value- کو square طاقت میں اٹھایا جاتا ہے اور پھر فوراً List میں شامل کیا جاتا ہے۔

آپ ان دونوں طریقوں کو استعمال کرتے ہوئے زیادہ پیچیدہ List بنا سکتے ہیں۔ کبھی کبھار عارضی variable کا استعمال آپ کے code کو پڑھنے میں آسان بناتا ہے؛ اور کبھی کبھار یہ code کو غیر ضروری طور پر طویل بنا دیتا ہے۔ سب

سے پہلے code لکھنے پر توجہ مرکوز کریں جو آپ کو واضح طور پر سمجھ آتا ہو، اور پھر اپنے code کا جائزہ لیتے ہوئے مزید مؤثر طریقے تلاش کریں۔

نمبرز کی List کے ساتھ سادہ اعداد و شمار

Python کے کچھ فنکشنز نمبرز کی وں List کے ساتھ کام کرتے وقت مددگار ثابت ہوتے ہیں۔ مثال کے طور پر، آپ آسانی سے کسی List کا کم از کم، زیادہ سے زیادہ اور مجموعہ معلوم کر سکتے ہیں:

```
digits = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
print(min(digits))  % 0
print(max(digits))  % 9
print(sum(digits))  % 45
```

نوٹ: اس سیکشن میں دی گئی مثالیں چھوٹی وں List پر مبنی ہیں جو صفحے پر آسانی سے فٹ ہو جاتی ہیں۔ یہ بالکل اتنی ہی اچھی طرح کام کریں گی اگر آپ کی List میں لاکھوں یا اس سے زیادہ نمبر ہوں۔

Dr. Muhammad Siddique 03006329919

3.2 List کمپریہنشنز

پہلے statement-کردہ طریقہ کار میں List-squares بنانے کے لئے تین یا چار لائنوں کا code استعمال کیا گیا تھا۔ ایک List کمپریہنشن آپ کو یہ ہی List صرف ایک لائن کے code میں بنانے کی اجازت دیتی ہے۔ ایک List کمپریہنشن فور Loop اور نئے عناصر بنانے کو ایک لائن میں یکجا کرتا ہے، اور ہر نئے element کو خود بخود List میں شامل کر دیتا ہے۔ List کمپریہنشنز ہمیشہ ابتدائی سیکھنے والوں کو نہیں دکھائی جاتیں، مگر میں نے انہیں یہاں شامل کیا ہے کیونکہ آپ انہیں ممکنہ طور پر دوسرے لوگوں کے code دیکھنا شروع کرتے ہی دیکھیں گے۔

اگلا مثال وہی numbersquareList بناتی ہے جو آپ نے پہلے دیکھی تھی مگر یہ List کمپریہنشن کا استعمال کرتی ہے:

```
squares = [value**2 for value in range(1, 11)]
print(squares)
```

اس سنٹیکس کو استعمال کرنے کے لیے، سب سے پہلے List کا ایک وضاحتی نام منتخب کریں، جیسے کہ squares۔ پھر square بریکٹ کھولیں اور اس اظہار کو statement- کریں جو آپ نئے List میں محفوظ کرنا چاہتے ہیں۔ اس مثال میں اظہار value**2 ہے، جو value- کو دوسرے درجے تک بڑھاتا ہے۔ پھر ایک فور Loop لکھیں جو وہ نمبرز پیدا کرے جنہیں آپ اظہار میں ڈالنا چاہتے ہیں، اور square بریکٹ کو بند کریں۔ اس مثال میں فور Loop value for in range(1,11) ہے، جو 1 سے 10 تک کے نمبرز کو اظہار value**2 میں داخل کرتا ہے۔ نوٹ کریں کہ فور اسٹیٹمنٹ کے آخر میں کوئی کالن استعمال نہیں کی جاتی۔

نتیجہ وہی List-square-numbers ہے جو آپ نے پہلے دیکھی تھی:

$$[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]$$

اپنی List کمپریہنشنز لکھنے میں excercise درکار ہوتی ہے، مگر جب آپ عام سی List بنانے میں ماہر ہو جائیں گے تو آپ انہیں قابل value پائیں گے۔ جب آپ تین یا چار لائنوں کے code سے List بناتے ہیں اور یہ محسوس ہوتا ہے کہ یہ بار بار دہرایا جا رہا ہے تو اپنے List کمپریہنشنز لکھنے پر غور کریں۔

Excercise

1. بیس تک گنتی کرنا: فور Loop کا استعمال کرتے ہوئے 1 سے 20 تک کے نمبرز پرنٹ کریں۔

2. ایک ملین: ایک سے ایک ملین تک کے نمبرز کی List بنائیں، اور پھر فور Loop کا استعمال کرتے ہوئے ان نمبرز کو پرنٹ کریں۔ (اگر output بہت زیادہ وقت لے رہا ہے تو اسے CTRL-C دبا کر روکیں یا output ونڈو بند کریں۔)

3. ایک ملین کا مجموعہ: ایک سے ایک ملین تک کے نمبرز کی List بنائیں، اور پھر min() اور max() کا استعمال کریں تاکہ یہ یقینی بنائیں کہ آپ کی List واقعی ایک سے شروع ہو رہی ہے اور ایک ملین پر ختم ہو رہی ہے۔ اس کے علاوہ، sum() فنکشن کا استعمال کریں تاکہ آپ دیکھ سکیں کہ Python ایک ملین نمبرز کو کتنی تیزی سے جمع کر سکتی ہے۔

Dr. Muhammad Siddique 03006329919

4. طاق نمبرز: range() فنکشن کے تیسرے آرگومنٹ کا استعمال کرتے ہوئے 1 سے 20 تک کے طاق نمبرز کی List بنائیں۔ ہر نمبر کو پرنٹ کرنے کے لئے فور Loop کا استعمال کریں۔

5. تین کے ضرب: 3 سے 30 تک کے تین کے ضرب کی List بنائیں۔ اپنے List میں موجود نمبرز کو پرنٹ کرنے کے لئے فور Loop کا استعمال کریں۔

6. کیوبس: ایک نمبر کو تیسرے درجے تک بڑھانے کو کیوب کہتے ہیں۔ مثال کے طور پر، 2 کا کیوب Python میں 2**3 کے طور پر لکھا جاتا ہے۔ پہلے 10 کیوبز کی List بنائیں (یعنی 1 سے 10 تک کے ہر عدد کا کیوب)، اور پھر فور Loop کا استعمال کرتے ہوئے ہر کیوب کی value پرنٹ کریں۔

7. کیوب کمپریہنشن: ایک List کمپریہنشن کا استعمال کرتے ہوئے پہلے 10 کیوبز کی List بنائیں۔

باب 3 میں آپ نے سیکھا تھا کہ List میں موجود انفرادی عناصر تک کیسے access حاصل کی جائے، اور اس باب میں آپ یہ سیکھ رہے ہیں کہ List میں موجود تمام عناصر کے ساتھ کیسے کام کیا جائے۔ آپ List میں موجود ایک مخصوص گروپ کے ساتھ بھی کام کر سکتے ہیں، جسے Python میں سلیس کہا جاتا ہے۔

List کو سلائس کرنا

سلائس بنانے کے لیے، آپ اس List کے پہلے اور آخری element کا انڈیکس بتاتے ہیں جس کے ساتھ آپ کام کرنا چاہتے ہیں۔ جیسے range () فنکشن کے ساتھ ہوتا ہے، Python دوسرے انڈیکس سے ایک آئٹم پہلے رک جاتا ہے جو آپ نے مخصوص کیا ہو۔ List میں پہلے تین عناصر کو output کرنے کے لیے، آپ انڈیکس 0 سے 3 تک درخواست کریں گے، جو عناصر 0، 1، اور 2 واپس کرے گا۔ یہاں ایک مثال ہے جو ٹیم کے کھلاڑیوں کی List پر مشتمل ہے:

```
1 players = ['charles', 'martina', 'michael', 'florence', 'eli']
2 print(players[0:3])
```

یہ st-Li-code کا ایک سلائس پرنٹ کرتا ہے۔ output List کی ساخت کو برقرار رکھتا ہے اور List کے پہلے تین کھلاڑیوں کو شامل کرتا ہے:

```
1 ['charles', 'martina', 'michael']
```

Dr. Muhammad Siddique 03006329919

آپ List کا کوئی بھی سب سیٹ پیدا کر سکتے ہیں۔ مثال کے طور پر، اگر آپ List میں دوسرے، تیسرے، اور چوتھے آئٹمز چاہتے ہیں، تو آپ سلائس کو انڈیکس 1 سے شروع کریں گے اور انڈیکس 4 پر ختم کریں گے:

```
1 players = ['charles', 'martina', 'michael', 'florence', 'eli']
2 print(players[1:4])
```

اس بار سلائس 'martina' سے شروع ہوتا ہے اور 'florence' پر ختم ہوتا ہے:

```
1 ['martina', 'michael', 'florence']
```

اگر آپ سلائس میں پہلے انڈیکس کو حذف کرتے ہیں، تو Python خود بخود آپ کا سلائس List کے آغاز سے شروع کرتا ہے:

```
1 players = ['charles', 'martina', 'michael', 'florence', 'eli']
2 print(players[:4])
```

بغیر کسی ابتدائی انڈیکس کے، Python List کے آغاز سے شروع ہوتا ہے:

```
1 ['charles', 'martina', 'michael', 'florence']
```

اسی طرح کا syntax اس صورت میں کام کرتا ہے جب آپ List کے آخر کو شامل کرنا چاہتے ہیں۔ مثال کے طور پر، اگر آپ تیسرے آئٹم سے لے کر آخری آئٹم تک تمام آئٹمز چاہتے ہیں، تو آپ انڈیکس 2 سے شروع کریں گے اور دوسرے انڈیکس کو حذف کر دیں گے:

```
1 players = ['charles', 'martina', 'michael', 'florence', 'eli']
2 print(players[2:])
```

Python تیسرے آئٹم سے لے کر List کے آخر تک تمام آئٹمز واپس کرتا ہے:

```
['michael', 'florence', 'eli']
```

یہ syntax آپ کو List کے کسی بھی نقطے سے لے کر آخر تک تمام عناصر output کرنے کی اجازت دیتا ہے، چاہے List کی لمبائی کچھ بھی ہو۔ یاد رکھیں کہ منفی انڈیکس List کے آخر سے ایک مخصوص فاصلے پر موجود element کو واپس کرتا ہے؛ اس لیے آپ List کے آخر سے کسی بھی سلائس کو output کر سکتے ہیں۔ مثال کے طور پر، اگر ہم List کے آخری تین کھلاڑیوں کو output کرنا چاہتے ہیں، تو ہم سلائس players[-3:] استعمال کر سکتے ہیں:

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
print(players[-3:])
```

یہ آخری تین کھلاڑیوں کے نام پرنٹ کرتا ہے اور جب کھلاڑیوں کی List سائز میں تبدیلی کرتی ہے، تو یہ کام کرتا رہتا ہے۔

نوٹ: آپ بریکٹس میں ایک تیسری value-بھی شامل کر سکتے ہیں جو سلائس کے درمیان کتنے آئٹمز کو چھوڑنا ہے، یہ Python کو بتاتا ہے۔

سلائس کے ذریعے Loop کرنا

Dr. Muhammad Siddique 03006329919

آپ اگر چاہتے ہیں کہ List کے ایک سب سیٹ پر Loop کریں تو آپ سلائس کا استعمال کر سکتے ہیں۔ اگلی مثال میں، ہم پہلے تین کھلاڑیوں پر Loop کرتے ہیں اور ان کے نام ایک سادہ راسٹر کے حصے کے طور پر پرنٹ کرتے ہیں:

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
print("Here are the first three players on my team:")
for player in players[:3]:
    print(player.-title())
```

Python تمام کھلاڑیوں کی List پر Loop کرنے کے بجائے صرف پہلے تین ناموں پر Loop کرتا ہے:

```
Here are the first three players on my team:
Charles
Martina
Michael
```

سلائس مختلف Conditions میں بہت مفید ہیں۔ مثال کے طور پر، جب آپ کھیل بنا رہے ہوں، تو آپ ہر بار کسی کھلاڑی کا آخری سکور List میں شامل کر سکتے ہیں۔ پھر آپ List کو کم کرنے کے بعد، سب سے پہلے تین سکوروں کے لیے ایک سلائس لے سکتے ہیں۔ جب آپ ڈیٹا کے ساتھ کام کر رہے ہوں تو آپ ڈیٹا کو مخصوص سائز کے ٹکڑوں میں پروسیس کرنے کے لیے سلائس کا استعمال کر سکتے ہیں۔ یا جب آپ ویب ایپلی کیشن بنا رہے ہوں، تو آپ سلائس کا استعمال صفحات کی ایک سیریز میں معلومات دکھانے کے لیے کر سکتے ہیں، جس میں ہر صفحے پر مناسب مقدار میں معلومات ہو۔

List کو نقل کرنا

اکثر آپ ایک موجودہ List کے ساتھ شروع کرنا چاہتے ہیں اور پہلی List کی بنیاد پر ایک نئی List بنانا چاہتے ہیں۔ آیئے

دیکھتے ہیں کہ List کو نقل کرنا کس طرح کام کرتا ہے اور ایک ایسی صورتحال پر نظر ڈالتے ہیں جس میں List کو نقل کرنا مفید ثابت ہوتا ہے۔ List کو نقل کرنے کے لیے، آپ ایک سلائس بنا سکتے ہیں جو پوری اصل List کو شامل کرتا ہے اور پہلے اور دوسرے انڈیکس کو حذف کر دیتا ہے ([:])۔ اس سے Python کو یہ پیغام ملتا ہے کہ یہ سلائس پہلی چیز سے شروع ہو کر آخری چیز تک پہنچے، اور پوری List کی نقل تیار کرے۔ مثال کے طور پر، فرض کریں ہمارے پاس پسندیدہ کھانوں کی ایک List ہے اور ہم ایک علیحدہ List بنانا چاہتے ہیں جو ہمارے دوست کو پسند ہو۔ ہمارا دوست اب تک ہماری List میں موجود تمام چیزوں کو پسند کرتا ہے، اس لیے ہم ان کی List ہماری List کو نقل کر کے بنا سکتے ہیں:

```
my_foods = ['pizza', 'falafel', 'carrot cake']
friend_foods = my_foods[:]
print("My favorite foods are:")
print(my_foods)
print("\nMy friend's favorite foods are:")
print(friend_foods)
```

پہلے ہم اپنی پسندیدہ کھانوں کی List بناتے ہیں جسے my_foods کہا جاتا ہے۔ پھر ہم ایک نئی List بناتے ہیں جسے friend_foods کہا جاتا ہے۔ ہم my_foods کا ایک سلائس بنا کر اس کی نقل بناتے ہیں، اور اس نقل کو friend_foods میں محفوظ کرتے ہیں۔ جب ہم ہر List کو پرنٹ کرتے ہیں تو ہم دیکھتے ہیں کہ دونوں وں List میں وہی کھانے شامل ہیں:

Dr. Muhammad Siddique 03006329969

```
My favorite foods are:
['pizza', 'falafel', 'carrot cake']
My friend's favorite foods are:
['pizza', 'falafel', 'carrot cake']
```

یہ ثابت کرنے کے لیے کہ ہمارے پاس واقعی دو علیحدہ ہیں List ہیں، ہم ہر List میں نیا کھانا شامل کرتے ہیں اور دکھاتے ہیں کہ ہر List صحیح person کے پسندیدہ کھانوں کا حساب رکھتی ہے:

```
my_foods = ['pizza', 'falafel', 'carrot cake']
friend_foods = my_foods[:]
my_foods.append('cannoli')
friend_foods.append('ice cream')
print("My favorite foods are:")
print(my_foods)
print("\nMy friend's favorite foods are:")
print(friend_foods)
```

ہم my_foods میں موجود اصل elements کو نئی List friend_foods میں نقل کرتے ہیں جیسے ہم نے پچھلی مثال میں کیا تھا۔ اس کے بعد ہم ہر List میں نیا کھانا شامل کرتے ہیں: ہم my_foods میں 'cannoli' شامل کرتے ہیں اور friend_foods میں "ice cream" شامل کرتے ہیں۔ پھر ہم دونوں List کو پرنٹ کرتے ہیں تاکہ یہ دیکھ سکیں کہ ہر کھانا متعلقہ List میں موجود ہے:

```
My favorite foods are:
['pizza', 'falafel', 'carrot cake', 'cannoli']
My friend's favorite foods are:
['pizza', 'falafel', 'carrot cake', 'ice cream']
```

output سے ظاہر ہوتا ہے کہ 'cannoli' اب ہماری پسندیدہ کھانوں کی List میں موجود ہے لیکن "ice cream" نہیں ہے۔ ہم دیکھ سکتے ہیں کہ "ice cream" اب ہمارے دوست کی List میں موجود ہے لیکن 'cannoli' نہیں ہے۔
اگر ہم نے friend_foods کو my_foods کے برابر بنادیا ہوتا، تو ہم دو علیحدہ List پیدا نہیں کرتے۔

Dr. Muhammad Siddique 03006329919

Tuples

یںListان elements کے مجموعوں کو store کرنے کے لیے بہترین کام کرتی ہیں جو ایک پروگرام کی زندگی کے دوران تبدیل ہو سکتی ہیں۔ وںList کو تبدیل کرنے کی صلاحیت اس وقت خاص طور پر اہم ہوتی ہے جب آپ ویب سائٹ پر users کی List یا کھیل میں کرداروں کی List کے ساتھ کام کر رہے ہوں۔ تاہم، بعض اوقات آپ کو ایسی elements کی List بنانے کی ضرورت ہوتی ہے جو تبدیل نہ ہو سکیں۔ زTuple آپ کو یہی کرنے کی اجازت دیتے ہیں۔ Python میں جن values کو تبدیل نہیں کیا جا سکتا انہیں "غیر قابل تبدیلی" (immutable) کہا جاتا ہے، اور ایک غیر قابل تبدیلی List کو Tuple کہا جاتا ہے۔

Tuple کی declare

ایک Tuple List کی طرح نظر آتا ہے، سوائے اس کے کہ آپ اس میں square قوسین کی بجائے گول قوسین استعمال کرتے ہیں۔ جب آپ ایک Tuple کی declare کر لیتے ہیں، تو آپ ہر شے کے انڈیکس کو استعمال کرتے ہوئے انفرادی عناصر تک access حاصل کر سکتے ہیں، جیسے کہ آپ List کے عناصر کے ساتھ کرتے ہیں۔

مثال کے طور پر، اگر ہمارے پاس ایک مستطیل ہے جس کا سائز ہمیشہ ایک خاص مقدار ہونا چاہیے، تو ہم اس کے سائز کو تبدیل ہونے سے روکنے کے لیے اس کے ابعاد کو Tuple میں رکھ سکتے ہیں:

Dr. Muhammad Siddique 03008329329

```
dimensions.py
dimensions = (200, 50)
print(dimensions[0])
print(dimensions[1])
```

ہم نے Tuple-dimensions کو گول قوسین () استعمال کر کے declare کیا۔ پھر ہم ہر element کو انفرادی طور پر پرنٹ کرتے ہیں، جیسے ہم List کے عناصر کو تک access حاصل کرنے کے لیے کرتے ہیں۔

```
200
50
```

اب دیکھتے ہیں کہ اگر ہم Tuple-dimensions میں سے کسی element کو تبدیل کرنے کی کوشش کریں تو کیا ہوتا ہے:

```
dimensions = (200, 50)
dimensions[0] = 250
```

یہ code پہلے element کی value- کو تبدیل کرنے کی کوشش کرتا ہے، لیکن Python ایک TypeError واپس کرتا ہے۔ چونکہ ہم Tuple کو تبدیل کرنے کی کوشش کر رہے ہیں، جو کہ اس قسم کی elements کے لیے ممکن نہیں ہے، Python ہمیں بتاتا ہے کہ ہم Tuple میں کسی element کو نیا assignvalue نہیں کر سکتے:

```
Traceback (most recent call last):
File "dimensions.py", line 2, in <module>
dimensions[0] = 250
```

```
TypeError: 'tuple' object does not support item assignment
```

یہ فائدہ مند ہے کیونکہ ہم چاہتے ہیں کہ جب کوئی code کو تبدیل کرنے کی کوشش کرے تو Python ایک غلطی پیدا کرے۔

نوٹ: Tuple کو تکنیکی طور پر () کے ذریعے declare کیا جاتا ہے؛ گول قوسین ان کو زیادہ صاف اور قابل پڑھائی بناتے ہیں۔ اگر آپ ایک element کے ساتھ declare Tuple کرنا چاہتے ہیں تو آپ کو ایک اضافی () شامل کرنا ضروری ہے:

```
my_t = (3,)
```

ایک element کے ساتھ Tuple بنانا اکثر معقول نہیں ہوتا، لیکن یہ اس وقت ہو سکتا ہے جب Tuple خود بخود پیدا کیے جائیں۔

Tuple میں تمام-values کے ذریعے Loop چلانا

آپ Tuple میں تمام-values کے ذریعے Loop چلا سکتے ہیں، جیسے آپ List کے ساتھ کرتے ہیں:

```
dimensions = (200, 50)
for dimension in dimensions:
    print(dimension)
```

Python-Tuple میں تمام عناصر واپس کرتا ہے، جیسے وہ List کے لیے کرتا:

```
200
50
```

Dr. Muhammad Siddique 03006329919

Tuple پر دوبارہ لکھنا

اگرچہ آپ Tuple کو تبدیل نہیں کر سکتے، آپ ایک نئی-value کو اس variable میں assign کر سکتے ہیں جو Tuple کی نمائندگی کرتا ہے۔ مثال کے طور پر، اگر ہم اس کے declare تبدیل کرنا چاہتے ہیں، تو ہم پورے Tuple کو دوبارہ declare کر سکتے ہیں:

```
dimensions = (200, 50)
print("Original dimensions:")
for dimension in dimensions:
    print(dimension)
dimensions = (400, 100)
print("\nModified dimensions:")
for dimension in dimensions:
    print(dimension)
```

پہلے چار لائنیں اصل Tuple کو declare کرتی ہیں اور ابتدائی ابعاد کو پرنٹ کرتی ہیں۔ پھر ہم variable-dimensions کے ساتھ ایک نیا Tuple منسلک کرتے ہیں اور نئی value-ں پرنٹ کرتے ہیں۔ اس بار Python کوئی غلطی نہیں دیتا، کیونکہ variable کو دوبارہ assign کرنا درست ہے:

```
Original dimensions:
200
50
Modified dimensions:
400
100
```

وں List کے مقابلے میں، زTuple سادہ ڈیٹا ڈھانچے ہیں۔ انہیں اس وقت استعمال کریں جب آپ ایسی value-وں کا مجموعہ store کرنا چاہتے ہیں جنہیں پروگرام کی زندگی کے دوران تبدیل نہیں کیا جانا چاہیے۔

Excercise

بفیٹ: ایک بفیٹ طرز کا ریسٹورنٹ صرف پانچ بنیادی کھانے پیش کرتا ہے۔ پانچ سادہ کھانے سوچیں، اور انہیں ایک Tuple میں store کریں۔

- ہر کھانے کو پرنٹ کرنے کے لیے ایک for Loop استعمال کریں جو ریسٹورنٹ پیش کرتا ہے۔
- ایک شے کو تبدیل کرنے کی کوشش کریں، اور یہ یقینی بنائیں کہ Python تبدیلی کو مسترد کرتا ہے۔
- ریسٹورنٹ اپنے مینو کو تبدیل کرتا ہے، دو elements کو مختلف کھانوں سے تبدیل کرتا ہے۔ ایک لائن شامل کریں جو Tuple کو دوبارہ لکھتی ہے، اور پھر ایک for Loop کا استعمال کرتے ہوئے تبدیل شدہ مینو میں سے ہر شے کو پرنٹ کریں۔

Dr. Muhammad Siddique 03006329919

آپ کے code کی styling

اب جب کہ آپ طویل پروگرام لکھ رہے ہیں، یہ ایک اچھا خیال ہے کہ آپ اپنے code کو مستقل طور پر Style کریں۔ اپنے code کو جتنا ممکن ہو آسان اور پڑھنے میں آسان بنائیں۔ آسان پڑھنے کے قابل code لکھنا آپ کو اپنے پروگرامز کی کارکردگی کو ٹریک کرنے میں مدد دیتا ہے اور دوسروں کو آپ کے code کو سمجھنے میں بھی مدد فراہم کرتا ہے۔

Python پروگرامرز نے ایک تعداد نگ Style کی کنونشنز پر اتفاق کیا ہے تاکہ یہ یقینی بنایا جا سکے کہ سب کا code تقریباً ایک ہی طریقے سے منظم ہو۔ ایک بار جب آپ صاف Python-code لکھنے کا طریقہ سیکھ لیں گے، تو آپ دوسرے کسی بھی person-کے code کا عمومی ڈھانچہ سمجھنے کے قابل ہوں گے، بشرطیکہ وہ یہی رہنما خطوط استعمال کریں۔ اگر آپ کسی وقت پروفیشنل پروگرامر بننے کی خواہش رکھتے ہیں، تو آپ کو جتنی جلدی ہو سکے یہ رہنما اصول اپنانا شروع کر دینا چاہیے تاکہ اچھے عادات کو فروغ دیا جا سکے۔

Style گائیڈ

جب کوئی Pythonperson زبان میں تبدیلی کرنا چاہتا ہے تو وہ ایک Python اینہنسمنٹ پروپوزل (PEP) لکھتا ہے۔ سب سے پرانی PEPs میں سے ایک 8PEP ہے، جو Python پروگرامرز کو ان کے code کی نگ Style کے بارے میں ہدایات فراہم کرتا ہے۔ 8PEP کافی طویل ہے، لیکن اس کا بیشتر حصہ پیچیدہ نگ code ڈھانچوں سے متعلق ہے جو آپ نے ابھی تک نہیں دیکھا ہے۔

Style-Python گائیڈ اس سمجھ کے ساتھ لکھا گیا تھا کہ code کو زیادہ پڑھا جاتا ہے بنسبت اس کے لکھنے کے۔ آپ اپنا code ایک بار لکھیں گے اور پھر اسے پڑھنا شروع کریں گے جب آپ ڈی بگنگ کر رہے ہوں گے۔ جب آپ کسی پروگرام میں خصوصیات شامل کرتے ہیں، تو آپ اپنا code زیادہ وقت پڑھیں گے۔ جب آپ اپنے code کو دوسرے پروگرامرز کے ساتھ شیئر کرتے ہیں، تو وہ بھی آپ کے code کو پڑھیں گے۔

جب آپ code لکھنے کے لیے انتخاب کرتے ہیں تو Python پروگرامرز ہمیشہ آپ کو ایسا code لکھنے کی ترغیب دیں گے جو پڑھنے میں آسان ہو، نہ کہ ایسا جو لکھنے میں آسان ہو۔ ذیل میں دی گئی رہنما خطوط آپ کو شروع سے ہی صاف code لکھنے میں مدد کریں گی۔

Dr. Muhammad Siddique 0300632919

Indentation

8PEP کی سفارش ہے کہ آپ ہر Indentation سطح کے لیے چار اسپیسز استعمال کریں۔ چار اسپیسز کا استعمال پڑھنے کی صلاحیت کو بہتر بناتا ہے اور ہر لائن پر Indentation کی متعدد سطحوں کے لیے جگہ چھوڑتا ہے۔

-word-کے پروسیسنگ دستاویزات میں لوگ عموماً ٹیبز استعمال کرتے ہیں، لیکن Python انٹرپرٹر اس وقت الجھ جاتا ہے جب ٹیبز اور اسپیسز کو ملا دیا جاتا ہے۔ ہر ٹیکسٹ ایڈیٹر ایک سیٹنگ فراہم کرتا ہے جو آپ کو ٹیب کی کنجی استعمال کرنے کی اجازت دیتی ہے، لیکن پھر ہر ٹیب کو اسپیسز کے مخصوص سیٹ میں تبدیل کر دیتی ہے۔ آپ کو یقینی طور پر اپنے ٹیب کی کنجی کا استعمال کرنا چاہیے، لیکن اس بات کا بھی خیال رکھیں کہ آپ کا ایڈیٹر اسپیسز کو ٹیبز میں تبدیل کرنے کے لیے سیٹ ہو۔

ٹیبز اور اسپیسز کو آپ کی فائل میں ملانے سے مسائل پیدا ہو سکتے ہیں جو تشخیص کرنا بہت مشکل ہوتے ہیں۔ اگر آپ کو لگتا ہے کہ آپ کی فائل میں ٹیبز اور اسپیسز کا مکس ہے، تو آپ اکثر ایڈیٹرز میں تمام ٹیبز کو اسپیسز میں تبدیل کر سکتے ہیں۔

لائین کی لمبائی

بہت سے Python پروگرامرز تجویز کرتے ہیں کہ ہر لائن کی لمبائی 80 حروف سے کم ہونی چاہیے۔ تاریخی طور پر، یہ رہنمائی اس وجہ سے تیار ہوئی تھی کیونکہ بیشتر کمپیوٹرز ایک واحد لائن میں صرف 79 حروف ہی رکھ سکتے تھے۔

اس وقت، لوگ اسکرین پر زیادہ لمبے لائنیں فٹ کر سکتے ہیں، لیکن دیگر وجوہات بھی ہیں جن کے لیے 79 حروف کی لائن کی لمبائی پر عمل کرنا ضروری ہے۔

پروفیشنل پروگرامرز اکثر ایک ہی اسکرین پر کئی فائلیں کھولتے ہیں، اور معیار لائن کی لمبائی ان کو اس بات کی اجازت

دیتی ہے کہ وہ اسکرین پر کھلی ہوئی دو یا تین فائلوں کی پوری لائنیں دیکھ سکیں۔ 8PEP یہ بھی تجویز کرتا ہے کہ آپ اپنے تمام تبصروں کی لمبائی کو 72 حروف فی لائن تک محدود رکھیں۔

Empty لائنیں

آپ اپنے پروگرام کے مختلف حصوں کو بصری طور پر گروپ کرنے کے لیے Empty لائنیں استعمال کر سکتے ہیں۔ آپ کو Empty لائنیں استعمال کرنی چاہیے تاکہ آپ کی فائلیں منظم ہوں، لیکن اس کا ضرورت سے زیادہ استعمال نہ کریں۔ مثال کے طور پر، اگر آپ کے پاس پانچ لائنوں کا code ہے جو ایک List بناتا ہے اور پھر تین لائنیں جو اس List کے ساتھ کچھ کرتی ہیں، تو ان دو حصوں کے درمیان ایک Empty لائن رکھنا مناسب ہوگا۔

دیگر Style رہنما اصول

8PEP میں کئی اضافی نک Style کی تجاویز بھی ہیں، لیکن ان میں سے زیادہ تر پیچیدہ پروگرامز سے متعلق ہیں جو آپ اس وقت لکھ رہے ہیں۔ جیسے جیسے آپ پیچیدہ Python ڈھانچوں کو سیکھیں گے، میں آپ کو 8PEP کی رہنما اصولوں کے متعلقہ حصے شیئر کروں گا۔

Dr. Muhammad Siddique 03006329919

خلاصہ

اس باب میں آپ نے سیکھا کہ کس طرح List میں موجود عناصر کے ساتھ مؤثر طریقے سے کام کیا جائے۔ آپ نے سیکھا کہ کس طرح For-Loop کا استعمال کرتے ہوئے List کے ذریعے کام کیا جائے، کس طرح پائیتھن پروگرام کو ڈھانچے دینے کے لئے Indentation کا استعمال کرتا ہے، اور کس طرح کچھ عام Indentation کی غلطیوں سے بچا جائے۔ آپ نے سادہ عددی یں List بنانا سیکھا، نیز کچھ آپریشنز جو آپ عددی وں List پر انجام دے سکتے ہیں۔ آپ نے سیکھا کہ کس طرح List کو ٹکڑوں میں تقسیم کیا جائے تاکہ آپ کسی مخصوص سب سیٹ کے ساتھ کام کر سکیں اور کس طرح List کو صحیح طریقے سے نقل کیا جائے۔ آپ نے ٹپلز کے بارے میں بھی سیکھا، جو ان values کے سیٹ کو تحفظ فراہم کرتے ہیں جو نہیں بدلے جانے چاہئیں، اور کس طرح اپنی بڑھتی ہوئی پیچیدہ code کو پڑھنے میں آسان بنانے کے لئے اسٹائل کیا جائے۔

باب 5 میں آپ سیکھیں گے کہ مختلف Conditions کے جواب میں مناسب طریقے سے کس طرح رد عمل ظاہر کیا جائے گا، اور آپ اگرات-statement کے ذریعے Testsconditional- کو آپس میں جوڑنا سیکھیں گے تاکہ آپ بالکل اسی قسم کی صورتحال یا معلومات کے مطابق رد عمل ظاہر کر سکیں جس کی آپ تلاش کر رہے ہیں۔ آپ اگر-statement ات کا استعمال کرتے ہوئے List کو Loop کرتے ہوئے مخصوص عناصر پر مخصوص کارروائیاں کرنے کا طریقہ بھی سیکھیں گے۔

declareکی IF Statement

پروگرامنگ اکثر مختلف Conditionsکا جائزہ لینے اور ان Conditionsکی بنیاد پر مناسب عمل کو انجام دینے پر مبنی ہوتی ہے۔ Python کا if اسٹیٹمنٹ آپ کو پروگرام کی موجودہ حالت کا جائزہ لینے اور اس حالت کے مطابق جواب دینے کی اجازت دیتا ہے۔

اس باب میں، آپ Conditional Tests لکھنے کا طریقہ سیکھیں گے، جو آپ کو کسی بھی مطلوبہ حالت کی جانچ کرنے کی اجازت دیتے ہیں۔ آپ سادہ if اسٹیٹمنٹ لکھنا سیکھیں گے، اور آپ ایک زیادہ پیچیدہ سلسلہ وار if اسٹیٹمنٹس بنانے کا طریقہ بھی سیکھیں گے تاکہ ان مخصوص Conditions کو پہچان سکیں جو آپ چاہتے ہیں۔ اس کے بعد آپ اس تصور کو lists پر لاگو کریں گے تاکہ آپ ایک for Loop لکھ سکیں جو List کے زیادہ تر عناصر کو ایک مخصوص طریقے سے سنبھالے، لیکن بعض عناصر کے لیے مختلف رویہ اختیار کرے۔

ایک سادہ مثال

Dr. Muhammad Siddique 03009229919

مندرجہ ذیل مثال دکھاتی ہے کہ if Test خاص Conditions کے لیے درست جواب دینے میں کس طرح مدد کرتے ہیں۔ فرض کریں کہ آپ کے پاس گاڑیوں کی ایک List ہے اور آپ ہر گاڑی کا نام پرنٹ کرنا چاہتے ہیں۔ گاڑیوں کے نام پروپر نام ہیں، لہذا زیادہ تر گاڑیوں کے نام کو ٹائٹل کیس میں پرنٹ کیا جانا چاہیے۔ تاہم، اگر 'bmw' value ہو تو اسے تمام بڑے حروف میں پرنٹ کرنا چاہیے۔ درج ذیل code گاڑیوں کے ناموں کی List پر ایک Loop چلاتا ہے اور 'bmw' کی تلاش کرتا ہے۔ جب بھی 'bmw' value ہو، اسے بڑے حروف میں پرنٹ کیا جاتا ہے:

```
cars.py
cars = ['audi', 'bmw', 'subaru', 'toyota']
for car in cars:
    if car == 'bmw':
        print(car.upper())
    else:
        print(car.-title())
```

اس مثال میں Loop پہلے یہ چیک کرتا ہے کہ موجودہ car کی 'bmw' value ہے یا نہیں۔ اگر ہے، تو value بڑے حروف میں پرنٹ کی جاتی ہے۔ اگر car کی 'bmw' value کے علاوہ کچھ اور ہو، تو یہ ٹائٹل کیس میں پرنٹ کی جاتی ہے:

```
Audi
BMW
Subaru
Toyota
```

یہ مثال کئی تصورات کو یکجا کرتی ہے جو آپ اس باب میں سیکھیں گے۔ آیئے ان Tests پر نظر ڈالتے ہیں جنہیں آپ اپنے پروگرام کی حالتوں کی جانچ کے لیے استعمال کر سکتے ہیں۔

Conditional Tests

ہر if اسٹیٹمنٹ کے مرکز میں ایک ایسا اظہار ہوتا ہے جسے True یا False کے طور پر جانچا جا سکتا ہے، اور اسے Conditional Test کہا جاتا ہے۔ Python-True اور False کی values استعمال کرتا ہے تاکہ یہ فیصلہ کیا جا سکے کہ if اسٹیٹمنٹ کے بعد code کو چلایا جائے یا نہیں۔ اگر Conditional-Test کا نتیجہ True ہو تو if Python اسٹیٹمنٹ کے بعد والے code کو چلاتا ہے۔ اگر نتیجہ False ہو، تو Python اس code کو نظر انداز کر دیتا ہے۔

برابری کے لیے جانچ

زیادہ تر Conditional-Test کسی variable کی موجودہ value کو کسی مطلوبہ value کے ساتھ موازنہ کرتے ہیں۔ سب سے آسان Conditional-Test یہ جانچتا ہے کہ آیا کسی variable کی value مطلوبہ value کے برابر ہے یا نہیں:

```
>>> car = 'bmw'
>>> car == 'bmw'
True
```

پہلی لائن میں ایک مساوی نشان (=) کے ذریعے car کی value کو 'bmw' پر سیٹ کیا جاتا ہے۔ اگلی لائن میں دو مساوی نشان (==) کے ذریعے یہ چیک کیا جاتا ہے کہ car کی value 'bmw' کے برابر ہے یا نہیں۔ یہ True واپس کرتا ہے کیونکہ دونوں values برابر ہیں۔ جب car کی value 'bmw' کے علاوہ کچھ اور ہو تو یہ Test False واپس کرتا ہے:

Dr. Muhammad Siddique 03006329919

```
>>> car = 'audi'
>>> car == 'bmw'
False
```

ایک سے زیادہ Conditions کی جانچ

آپ بعض اوقات ایک ہی وقت میں ایک سے زیادہ Conditions کی جانچ کرنا چاہیں گے۔ مثال کے طور پر، کبھی کبھار آپ کو دو True Conditions ہونے کی ضرورت ہو سکتی ہے تاکہ کوئی عمل کیا جا سکے۔ دیگر اوقات، آپ صرف ایک حالت True ہونے پر مطمئن ہو سکتے ہیں۔ ایسے مواقع پر and اور or-reserved--word- مددگار ثابت ہو سکتے ہیں۔

and کا استعمال کرتے ہوئے ایک سے زیادہ Conditions کی جانچ

دو Conditions کو ایک ساتھ True ہونے کی جانچ کرنے کے لیے، and-reserved--word- استعمال کریں تاکہ دونوں Conditional-Tests کو یکجا کیا جا سکے؛ اگر ہر Test کامیاب ہو، تو مجموعی اظہار True پر جانچتا ہے۔ اگر کسی ایک یا دونوں Tests میں ناکامی ہو تو یہ False پر جانچتا ہے۔

مثال کے طور پر، آپ یہ چیک کر سکتے ہیں کہ دو افراد 21 سال سے زیادہ عمر کے ہیں یا نہیں:

```
>>> age_0 = 22
>>> age_1 = 18
>>> age_0 >= 21 and age_1 >= 21
False
>>> age_1 = 22
>>> age_0 >= 21 and age_1 >= 21
True
```

سب سے پہلے، ہم دو عمریں age_0 اور age_1 متعین کرتے ہیں۔ پھر ہم جانچتے ہیں کہ دونوں عمریں 21 یا اس سے زیادہ ہیں یا نہیں۔ بائیں طرف کا Test کامیاب ہوتا ہے، لیکن دائیں طرف کا Test ناکام ہوتا ہے، اس لیے مجموعی اظہار False پر جانچتا ہے۔ اس کے بعد ہم age_1 کو 22 پر تبدیل کرتے ہیں۔ اب age_1 کی value 21 سے زیادہ ہے، اس لیے دونوں انفرادی Tests کامیاب ہوتے ہیں، اور مجموعی اظہار True پر جانچتا ہے۔

پڑھنے میں آسانی کے لیے، آپ انفرادی Tests کے ارد گرد قوسین استعمال کر سکتے ہیں، لیکن یہ لازمی نہیں ہیں۔ اگر آپ قوسین استعمال کرتے ہیں، تو آپ کا Test اس طرح دکھے گا:

```
(age_0 >= 21) and (age_1 >= 21)
```

Dr. Muhammad Siddique 03006329919

or کا استعمال کرتے ہوئے ایک سے زیادہ Conditions کی جانچ

-word--reserved-or بھی ایک سے زیادہ Conditions کی جانچ کرنے کی اجازت دیتا ہے، لیکن یہ تب کامیاب ہوتا ہے جب کسی ایک یا دونوں انفرادی Tests کامیاب ہوں۔ or اظہار صرف اس وقت ناکام ہوتا ہے جب دونوں انفرادی Tests ناکام ہوں۔

آیئے دوبارہ دو عمروں پر غور کرتے ہیں، لیکن اس بار ہم دیکھیں گے کہ آیا صرف ایک person 21 سال سے زیادہ عمر کا ہے:

```
>>> age_0 = 22
>>> age_1 = 18
>>> age_0 >= 21 or age_1 >= 21
True
>>> age_0 = 18
>>> age_0 >= 21 or age_1 >= 21
False
```

ہم دوبارہ دو عمر کے variables کے ساتھ شروع کرتے ہیں۔ چونکہ age_0 False، $Tests$، $Test$ 18 age_0 True، $Test_0$

یہ جانچنا کہ آیا کوئی value List میں موجود ہے

بعض اوقات یہ جانچنا ضروری ہوتا ہے کہ آیا List میں کوئی خاص value موجود ہے یا نہیں۔ مثال کے طور پر، آپ یہ چیک کرنا چاہیں گے کہ کسی نئی User کا نام پہلے سے موجود User ناموں کی List میں ہے یا نہیں۔ کسی میپنگ پروجیکٹ میں، آپ یہ دیکھنا چاہیں گے کہ آیا جمع کرائی گئی لوکیشن پہلے سے معلوم لوکیشنز کی List میں موجود ہے یا نہیں۔

یہ جاننے کے لیے کہ آیا کوئی مخصوص value پہلے ہی کسی List میں موجود ہے، word--reserved-in-استعمال کریں۔ آیئے ایک پیزا کی مثال پر غور کریں۔ ہم گاہک کے درخواست کردہ ٹاپنگز کی ایک List بنائیں گے اور پھر دیکھیں گے کہ مخصوص ٹاپنگز اس List میں موجود ہیں یا نہیں:

```
>>> requested_toppings = ['mushrooms', 'onions', 'pineapple']
>>> 'mushrooms' in requested_toppings
True
>>> 'pepperoni' in requested_toppings
False
```

یہ جانچنا کہ آیا کوئی Listvalue میں موجود نہیں ہے

کبھی کبھار یہ جاننا ضروری ہوتا ہے کہ آیا کوئی Listvalue میں موجود نہیں ہے۔ ایسی صورت میں reserved-in not--word-استعمال کریں۔ مثال کے طور پر، فرض کریں کہ users کی ایک List ہے جو فورم پر comments کرنے سے ممنوع ہیں۔ آپ چیک کر سکتے ہیں کہ آیا کسی User کو comment کرنے سے پہلے ممنوع کیا گیا ہے یا نہیں:

```
banned_users = ['andrew', 'carolina', 'david']
user = 'marie'

if user not in banned_users:
    print(f"{user.-title()}, you can post a response if you
    wish.")
```

Dr. Muhammad Siddique 03006329919

یہاں if اسٹیٹمنٹ واضح طور پر پڑھی جا سکتی ہے۔ اگر user کی banned_users-Listvalue میں نہیں ہے، تو Python-True واپس کرتا ہے اور اندرونی لائن چلاتا ہے۔ User-'marie'-List-banned_users-میں موجود نہیں ہیں، اس لیے وہ ایک پیغام دیکھتی ہیں جو انہیں comment کرنے کی دعوت دیتا ہے:

```
Marie, you can post a response if you wish.
```

Boolean-Expression

جب آپ پروگرامنگ کے بارے میں زیادہ جانتے ہیں، تو آپ کو Boolean""Expression کی اصطلاح سننے کو ملے گی۔ ایک BooleanExpression دراصل TestConditional کا دوسرا نام ہے۔ ایک Boolean-value True یا False ہوتی ہے، بالکل اسی طرح جیسے Conditional اظہار کی value کے جانچنے کے بعد ہوتی ہے۔

Boolean یں value اکثر مخصوص Conditions کا پتہ لگانے کے لیے استعمال کی جاتی ہیں، جیسے کہ آیا کوئی گیم چل رہا ہے یا آیا کوئی User ویب سائٹ پر مخصوص مواد میں ترمیم کر سکتا ہے:

```
game_active = True
can_edit = False
```

Boolean میں value پروگرام یا کسی خاص حالت کو مؤثر طریقے سے ٹریک کرنے کا ایک کارگر طریقہ فراہم کرتی ہیں جو آپ کے پروگرام میں اہم ہو سکتی ہے۔

Excercise

conditional-.1-5 ٹیسٹس: ایک سلسلہ وار conditional- ٹیسٹس لکھیں۔ ہر ٹیسٹ کی وضاحت کرنے والا statement- پرنٹ کریں اور ہر ٹیسٹ کے نتائج کی پیش گوئی کریں۔ آپ کا code کچھ اس طرح دکھنا چاہیے:

```
car = 'subaru'
print("Is car == 'subaru'? I predict True.")
print(car == 'subaru')
print("\nIs car == 'audi'? I predict False.")
print(car == 'audi')
```

- اپنے نتائج کو بغور دیکھیں اور یہ یقینی بنائیں کہ آپ سمجھتے ہیں کہ ہر لائن کیوں True یا False پر جانچتی ہے۔
- کم از کم 10 ٹیسٹس بنائیں۔ کم از کم 5 ٹیسٹس True کے نتائج دیں اور دیگر 5 ٹیسٹس False کے نتائج دیں۔

مزید conditional- ٹیسٹس: آپ کو صرف 10 ٹیسٹس تک محدود رہنے کی ضرورت نہیں ہے۔ اگر آپ مزید تقابل آزمانا چاہتے ہیں تو مزید ٹیسٹس لکھیں اور انہیں conditional_tests.py میں شامل کریں۔ درج ذیل میں سے ہر ایک کے لیے کم از کم ایک True اور ایک False نتیجہ شامل کریں:

Dr. Muhammad Siddique 03006329519

- سٹرنگز کے ساتھ برابری اور عدم برابری کے ٹیسٹس
- lower() method کا استعمال کرتے ہوئے ٹیسٹس
- عددی ٹیسٹس جن میں برابری، عدم برابری، بڑا ہونا اور چھوٹا ہونا، بڑا یا برابر ہونا، اور چھوٹا یا برابر ہونا شامل ہیں
- and اور or-reserved--word- استعمال کرتے ہوئے ٹیسٹس
- ٹیسٹ کریں کہ آیا کوئی آئٹم List میں موجود ہے
- ٹیسٹ کریں کہ آیا کوئی آئٹم List میں موجود نہیں ہے

4.2 if-Statements

جب آپ conditional- ٹیسٹ کو سمجھ لیتے ہیں، تو آپ if-statements لکھنا شروع کر سکتے ہیں۔ کئی مختلف اقسام کی if-statements موجود ہیں، اور آپ کا انتخاب ان میں سے کسی ایک کو استعمال کرنے کا انحصار اس بات پر ہے

کہ آپ کو کتنی شرائط کو جانچنا ہے۔ آپ نے conditional-ٹیسٹ کے بارے میں تبادلہ خیال میں کئی statements-if کے مثالیں دیکھی ہیں، لیکن اب آیئے اس موضوع میں مزید گہرائی میں جائیں۔

1.4.2 Statements-if-Simple

سب سے سادہ قسم کی statement-if ایک ٹیسٹ اور ایک عمل پر مشتمل ہوتی ہے:

```
if conditional_test:
    do something
```

آپ پہلی لائن میں کوئی بھی conditional-ٹیسٹ رکھ سکتے ہیں اور ٹیسٹ کے بعد والی انڈینٹڈ بلاک میں تقریباً کوئی بھی عمل کر سکتے ہیں۔ اگر conditional-ٹیسٹ True پر تشخیص ہوتا ہے تو Python اس کے بعد والا code چلائے گا۔ اگر ٹیسٹ False پر تشخیص ہوتا ہے تو Python اس کے بعد والا code نظر انداز کر دے گا۔

فرض کریں کہ ہمارے پاس کسی person کی عمر کی نمائندگی کرنے والی ایک variable ہے، اور ہم یہ جاننا چاہتے ہیں کہ کیا وہ person ووٹ دینے کے لیے کافی عمر کا ہے۔ ذیل کا code یہ جانچتا ہے کہ کیا وہ person ووٹ دے سکتا ہے:

```
age = 19
if age >= 18:
    print("You are old enough to vote!")
```

Dr. Muhammad Siddique 03006329919

Python یہ چیک کرتا ہے کہ کیا عمر کی value-18 سے بڑی یا برابر ہے۔ اگر ہے، تو Python انڈینٹڈ print() کال کو چلائے گا:

```
You are old enough to vote!
```

statements-if میں انڈینٹیشن وہی کردار ادا کرتا ہے جو loops for میں کرتا ہے۔ اگر ٹیسٹ کامیاب ہو جائے تو statement-if کے بعد تمام انڈینٹڈ لائنز چلیں گی، اور اگر ٹیسٹ کامیاب نہ ہو تو ان تمام لائنز کو نظر انداز کر دیا جائے گا۔

آپ statement-if کے بعد انڈینٹڈ بلاک میں جتنی چاہیں لائنز کا code شامل کر سکتے ہیں۔ اگر person ووٹ دینے کے لیے کافی عمر کا ہو تو ہم ایک اور لائن شامل کرتے ہیں، پوچھتے ہیں کہ کیا فرد نے ابھی تک ووٹ کے لیے رجسٹر کیا ہے:

```
age = 19
if age >= 18:
    print("You are old enough to vote!")
    print("Have you registered to vote yet?")
```

چونکہ conditional-ٹیسٹ کامیاب ہو چکا ہے، اور دونوں print() کالز انڈینٹڈ ہیں، تو دونوں لائنیں پرنٹ ہو جائیں گی:

```
You are old enough to vote!
Have you registered to vote yet?
```

اگر عمر کی value-18 سے کم ہو تو اس پروگرام کا کوئی output نہیں ہو گا۔

2.4.2 Statements-if-else

اکثر آپ چاہیں گے کہ جب conditional- ٹیسٹ کامیاب ہو تو ایک عمل کریں اور باقی تمام صورتوں میں ایک مختلف عمل کریں۔ Python کی if-else syntax اس بات کو ممکن بناتی ہے۔ ایک if-else بلاک ایک سادہ if-statement کی طرح ہوتا ہے، لیکن else-statement آپ کو ایک عمل یا کئی اعمال کو متعین کرنے کی اجازت دیتا ہے جو conditional- ٹیسٹ کے ناکام ہونے پر چلائے جاتے ہیں۔

ہم وہی پیغام دکھائیں گے جو پہلے تھا اگر person ووٹ دینے کے لیے کافی عمر کا ہے، لیکن اس بار ہم ایک پیغام بھی شامل کریں گے جو ان افراد کے لیے ہوگا جو ووٹ دینے کے لیے کافی عمر کے نہیں ہیں:

```
age = 17
if age >= 18:
    print("You are old enough to vote!")
    print("Have you registered to vote yet?")
else:
    print("Sorry, you are too young to vote.")
    print("Please register to vote as soon as you turn 18!")
```

اگر conditional- ٹیسٹ 1 کامیاب ہو جائے تو انڈینٹڈ print() کالز کا پہلا بلاک چلایا جائے گا۔ اگر ٹیسٹ False پر تشخیص ہو تو else بلاک 2 چلایا جائے گا۔ چونکہ اس بار عمر 18 سے کم ہے، conditional- ٹیسٹ ناکام ہو جاتا ہے اور else بلاک کا code چلتا ہے:

```
Sorry, you are too young to vote.
Please register to vote as soon as you turn 18!
```

Dr. Muhammad Siddique 03006329919

یہ code کام کرتا ہے کیونکہ اس میں صرف دو ممکنہ صورتوں کا جائزہ لیا جا سکتا ہے: ایک person یا تو ووٹ دینے کے لیے کافی عمر کا ہے یا نہیں۔ if-else ساخت ایسی صورتحال میں اچھی طرح کام کرتی ہے جب آپ چاہیں کہ Python ہمیشہ دو ممکنہ اعمال میں سے ایک کو انجام دے۔

5.2 if-elif-else چین

اکثر آپ کو دو سے زیادہ ممکنہ حالات کی جانچ کرنے کی ضرورت پیش آتی ہے، اور ان کو جانچنے کے لیے آپ Python کی if-elif-else syntax استعمال کر سکتے ہیں۔ Python اگر elif-else- چین میں صرف ایک بلاک کو ہی چلائے گا۔ یہ ہر شرطی ٹیسٹ کو ترتیب وار چلاتا ہے، جب تک کہ کوئی ٹیسٹ پاس نہ ہو جائے۔ جب کوئی ٹیسٹ پاس ہو جاتا ہے، تو اس کے بعد کا code چلایا جاتا ہے اور Python باقی تمام ٹیسٹوں کو نظر انداز کر دیتا ہے۔

کئی حقیقی دنیا کی صورتحالوں میں دو سے زیادہ ممکنہ حالات ہوتے ہیں۔ مثال کے طور پر، ایک تفریحی پارک کے بارے میں سوچیں جو مختلف عمر کے گروپوں کے لیے مختلف value- یں وصول کرتا ہے:

- 4 سال سے کم عمر کے لیے داخلہ مفت ہے۔
- 4 سے 18 سال کی عمر کے درمیان داخلہ 25 ڈالر ہے۔

- 18 سال یا اس سے زیادہ عمر کے افراد کے لیے داخلہ 40 ڈالر ہے۔

ہم ایک if-statement کیسے استعمال کر سکتے ہیں تاکہ کسی person کی داخلہ-value کا تعین کیا جا سکے؟ ذیل کا code کسی person کی عمر کے گروپ کی جانچ کرتا ہے اور پھر ایک داخلہ-value کا پیغام پرنٹ کرتا ہے:

```
amusement_park.py
age = 12
if age < 4:
    print("Your admission cost is $0.")
elif age < 18:
    print("Your admission cost is $25.")
else:
    print("Your admission cost is $40.")
```

یہ if ٹیسٹ 1 چیک کرتا ہے کہ آیا کوئی person 4 سال سے کم عمر کا ہے۔ جب یہ ٹیسٹ کامیاب ہوتا ہے، تو ایک مناسب پیغام پرنٹ کیا جاتا ہے اور Python باقی تمام ٹیسٹوں کو نظر انداز کر دیتا ہے۔ elif لائن 2 ایک اور if ٹیسٹ ہے جو صرف اس صورت میں چلتا ہے جب پچھلا ٹیسٹ ناکام ہو۔ اس وقت چین میں یہ جانچنا ضروری ہے کہ person کم از کم 4 سال کا ہے کیونکہ پہلا ٹیسٹ ناکام ہو چکا ہے۔ اگر person 18 سال سے کم ہے، تو ایک مناسب پیغام پرنٹ کیا جاتا ہے اور Python else-بلاک کو نظر انداز کر دیتا ہے۔ اگر if اور elif دونوں ٹیسٹ ناکام ہو جاتے ہیں، تو Python else-بلاک 3 کو چلائے گا۔

اس مثال میں if ٹیسٹ False 1 کے طور پر ریٹرن کرتا ہے، اس لیے اس کا code بلاک نہیں چلتا۔ تاہم، elif ٹیسٹ True کے طور پر ریٹرن کرتا ہے (12 سال 18 سے کم ہے) لہٰذا اس کا code چلایا جاتا ہے۔ نتیجہ یہ ہے:

```
Your admission cost is $25.
```

اگر عمر 17 سے کم ہو تو اس پروگرام میں کوئی output نہیں آئے گا۔

Dr. Muhammad Siddique 03006329919

6.2 if-elif-else بلاکس کا استعمال

آپ اپنے code میں جتنے چاہیں elif بلاکس استعمال کر سکتے ہیں۔ مثال کے طور پر، اگر تفریحی پارک سینئرز کے لیے رعایت پیش کرتا ہے تو آپ ایک اور شرطی ٹیسٹ شامل کر سکتے ہیں تاکہ یہ معلوم کیا جا سکے کہ آیا کوئی person سینئر ڈسکاؤنٹ کے لیے اہل ہے یا نہیں۔ فرض کریں کہ کوئی بھی person جو 65 سال یا اس سے زیادہ عمر کا ہے، وہ معمول کی داخلہ-value کے نصف یعنی 20 ڈالر ادا کرے گا:

```
age = 12
if age < 4:
    price = 0
elif age < 18:
    price = 25
elif age < 65:
    price = 40
else:
    price = 20
print(f"Your admission cost is ${price}.")
```

اس code کا بیشتر حصہ تبدیل نہیں ہوا۔ دوسرا elif بلاک اب اس بات کو چیک کرتا ہے کہ آیا کوئی 65 person سال سے کم ہے، اس سے پہلے کہ اسے مکمل داخلہ 40 value- ڈالر ملے۔ دھیان دیں کہ else بلاک میں value- کو 20 ڈالر میں تبدیل کیا گیا ہے کیونکہ صرف وہ لوگ جو 65 سال یا اس سے زیادہ عمر کے ہیں اس بلاک میں آئیں گے۔

7.2 else بلاک کو ختم کرنا

Python میں if-elif چین کے آخر میں else بلاک ضروری نہیں ہوتا۔ بعض اوقات، ایک else بلاک مفید ہوتا ہے، لیکن دوسری بار جب آپ کو کسی مخصوص شرط کے لیے اضافی statement-elif استعمال کرنے کی ضرورت ہوتی ہے، تو else بلاک کو چھوڑ دینا بہتر ہوتا ہے:

```
age = 12
if age < 4:
    price = 0
elif age < 18:
    price = 25
elif age < 65:
    price = 40
elif age >= 65:
    price = 20
print(f"Your admission cost is ${price}.")
```

Dr. Muhammad Siddique 03006329919

آخری elif- بلاک -- value کو 20 ڈالر پر سیٹ کرتا ہے جب 65 person- سال یا اس سے زیادہ عمر کا ہوتا ہے، جو کہ عام else بلاک سے زیادہ واضح ہے۔ اس تبدیلی کے ساتھ، ہر code بلاک کو چلانے کے لیے ایک مخصوص ٹیسٹ پاس کرنا ضروری ہے۔

8.2 if-elif-else

if-elif-else چین طاقتور ہوتا ہے، لیکن جب آپ کو ایک سے زیادہ حالات کی جانچ کرنی ہو تو اسے استعمال کرنا بہتر ہوتا ہے۔ جب Python کسی ٹیسٹ کو پاس کر لیتا ہے، تو وہ باقی ٹیسٹوں کو نظر انداز کر دیتا ہے۔ یہ رویہ فائدہ مند ہے کیونکہ یہ مؤثر ہے اور آپ کو ایک مخصوص حالت کی جانچ کرنے کی اجازت دیتا ہے۔

تاہم، بعض اوقات یہ ضروری ہوتا ہے کہ آپ تمام حالات کی جانچ کریں۔ اس صورت میں، آپ کو کئی سادہ if-statement استعمال کرنے چاہییں بغیر elif یا else بلاکس کے۔ یہ طریقہ کار اس وقت موزوں ہے جب ایک سے زیادہ حالتیں True ہو سکتی ہیں اور آپ ہر حالت کے لیے عمل کرنا چاہتے ہیں۔

9.2 List کے ساتھ statement-if کا استعمال

جب آپ List اور statement-if کو یکجا کرتے ہیں تو کچھ دلچسپ کام کر سکتے ہیں۔ آپ ان خاص value- کو دیکھ سکتے ہیں جنہیں List میں دیگر value- سے مختلف طریقے سے سلوک کرنا ضروری ہو۔ آپ بدلتی ہوئی شرائط جیسے کسی ریستوران میں شیفٹ کے دوران مخصوص elements کی دستیابی کو مؤثر طریقے سے منظم کر سکتے ہیں۔ آپ یہ بھی ثابت کر سکتے ہیں کہ آپ کا code تمام ممکنہ حالات میں جیسا آپ نے توقع کی تھی ویسا ہی کام کرتا ہے۔

10.2 خصوصی elements کی جانچ

اس باب کا آغاز ایک سادہ مثال سے ہوا تھا جس میں یہ دکھایا گیا تھا کہ bmw جیسے خصوصی value- کو کس طرح List میں دیگر value- وں سے مختلف فارمیٹ میں پرنٹ کیا جائے۔ اب جب کہ آپ کو conditional- ٹیسٹ اور statement-if ات کی بنیادی سمجھ حاصل ہو گئی ہے، آیئے مزید تفصیل سے دیکھیں کہ آپ List میں خاص value- وں کو کیسے دیکھ سکتے ہیں اور ان کو مناسب طریقے سے کیسے ہینڈل کر سکتے ہیں۔

آیئے پیزا بنانے والے مثال کو جاری رکھتے ہیں۔ پیزا بنانے کے دوران جب بھی کوئی ٹاپنگ شامل کی جاتی ہے، ایک پیغام ظاہر کرتا ہے۔ اس عمل کا code اس طرح سے لکھا جا سکتا ہے:

By Muhammad Siddique 03006329919

```
toppings.py
requested_toppings = ['mushrooms', 'green peppers', 'extra cheese']
for requested_topping in requested_toppings:
    print(f"Adding {requested_topping}.")
print("\nFinished making your pizza!")
```

output سیدھا سادہ ہے کیونکہ یہ code صرف ایک سادہ for Loop ہے:

```
Adding mushrooms.
Adding green peppers.
Adding extra cheese.
Finished making your pizza!
```

لیکن اگر سبز مرچ ختم ہو جائیں تو کیا ہو گا؟ ہم اس صورتحال کو مناسب طریقے سے ہینڈل کرنے کے لیے statement-if کا استعمال کر سکتے ہیں:

```
requested_toppings = ['mushrooms', 'green peppers', 'extra cheese']
for requested_topping in requested_toppings:
    if requested_topping == 'green peppers':
        print("Sorry, we are out of green peppers right now.")
    else:
        print(f"Adding {requested_topping}.")
print("\nFinished making your pizza!")
```

اس بار ہم ہر درخواست شدہ آئٹم کو پیزا میں شامل کرنے سے پہلے چیک کرتے ہیں۔statement-if چیک کرتا ہے کہ آیا سبز مرچ درکار ہیں۔ اگر ایسا ہے تو ہم ایک پیغام ظاہر کرتے ہیں جو اس بات کی وضاحت کرتا ہے کہ سبز مرچ دستیاب نہیں ہیں۔else بلاک اس بات کو یقینی بناتا ہے کہ تمام دیگر ٹاپنگز پیزا میں شامل کر لی جائیں۔output دکھاتا ہے کہ ہر درخواست شدہ ٹاپنگ کو مناسب طریقے سے ہینڈل کیا گیا:

```
Adding mushrooms.
Sorry, we are out of green peppers right now.
Adding extra cheese.
Finished making your pizza!
```

11.2 یہ جانچنا کہ Empty-List نہیں ہے

ہم نے اب تک جتنی بھی وں List کے ساتھ کام کیا ہے، ہم نے یہ سادہ مفروضہ کیا تھا کہ ہر List میں کم از کم ایک آئٹم ہوتا ہے۔ جلد ہی ہم users کو List میں محفوظ معلومات فراہم کرنے دیں گے، لہذا ہم یہ فرض نہیں کر سکیں گے کہ ہر بار جب Loop چلایا جائے تو List میں کوئی آئٹم موجود ہو۔ اس صورتحال میں، یہ مفید ہے کہ ہم یہ چیک کریں کہ آیا List Empty- ہے یا نہیں، اس سے پہلے کہ ہم Loopfor چلائیں۔

ایک مثال کے طور پر، آیئے ہم چیک کریں کہ آیا درخواست شدہ ٹاپنگز کی Empty-List ہے یا نہیں، اس سے پہلے کہ ہم پیزا بنائیں۔ اگر Empty-List ہو، تو ہم User سے پوچھیں گے اور اس بات کو یقینی بنائیں گے کہ وہ سادہ پیزا چاہتے ہیں۔ اگر Empty-List نہ ہو تو ہم پیزا بنائیں گے جیسا کہ ہم نے پچھلی مثالوں میں کیا:

Dr. Muhammad Siddique 03006509919

```
requested_toppings = []
if requested_toppings:
    for requested_topping in requested_toppings:
        print(f"Adding {requested_topping}.")
    print("\nFinished making your pizza!")
else:
    print("Are you sure you want a plain pizza?")
```

اس بار ہم درخواست شدہ ٹاپنگز کی ListEmpty کے ساتھ شروع کرتے ہیں۔ فوراً Loopfor میں جانے کی بجائے، ہم پہلے تیزی سے چیک کرتے ہیں۔ جب List کا نام statement-if میں استعمال کیا جاتا ہے، Python یہ واپس True کرتا ہے اگر List میں کم از کم ایک آئٹم ہو؛ ایک Empty-List False- کے طور پر تشخیص پاتی ہے۔ اگر درخواست شدہ ٹاپنگز conditional- ٹیسٹ پاس کرتی ہے تو ہم وہی Loopfor چلائیں گے جو ہم نے پچھلی مثال میں استعمال کیا۔ اگر conditional- ٹیسٹ ناکام ہو جاتا ہے تو ہم ایک پیغام پرنٹ کریں گے جس میں User سے پوچھا جائے گا کہ کیا وہ واقعی بغیر کسی ٹاپنگ کے سادہ پیزا چاہتے ہیں۔

اس معاملے میں EmptyList ہے، لہذا User سے output پوچھے گا:

```
Are you sure you want a plain pizza?
```

اگر EmptyList نہ ہو، تو output ہر درخواست شدہ ٹاپنگ کو پیزا میں شامل کرنے کو دکھائے گا۔

12.2 Lists-Multiple کا استعمال

لوگ کچھ بھی مانگیں گے، خاص طور پر جب پیزا ٹاپنگز کی بات آتی ہے۔ اگر ایک User اپنے پیزا پر فرانسیسی فرائز رکھنا چاہتا ہو تو کیا ہو گا؟ آپ وں List اور if ات-statement کا استعمال کر سکتے ہیں تاکہ یہ یقینی بن سکے کہ آپ کی input منطقی ہے اس سے پہلے کہ آپ اس پر عمل کریں۔

آیئے اس بات کا خیال رکھتے ہیں کہ ہم پیزا بنانے سے پہلے غیر معمولی ٹاپنگ کی درخواستوں کو دیکھیں۔ نیچے دی گئی مثال دووں List کو متعین کرتی ہے۔ پہلی List پیزیریا میں دستیاب ٹاپنگز کی List ہے، اور دوسری List وہ ٹاپنگز ہے جو User نے درخواست کی ہیں۔ اس بار، ہر درخواست شدہ آئٹم کو پیزا میں شامل کرنے سے پہلے دستیاب ٹاپنگز کی List سے چیک کیا جاتا ہے:

```
available_toppings = ['mushrooms', 'olives', 'green peppers', '
    pepperoni', 'pineapple', 'extra cheese']
requested_toppings = ['mushrooms', 'french fries', 'extra
    cheese']
for requested_topping in requested_toppings:
    if requested_topping in available_toppings:
        print(f"Adding {requested_topping}.")
    else:
        print(f"Sorry, we don't have {requested_topping}.")
print("\nFinished making your pizza!")
```

Dr. Muhammad Siddique 03006329919

سب سے پہلے، ہم اس پیزیریا میں دستیاب ٹاپنگز کی List متعین کرتے ہیں۔ اگر پیزیریا میں ٹاپنگز کا انتخاب مستحکم ہو تو یہ ایک ٹپول (tuple) ہو سکتی ہے۔ پھر ہم ایک List بناتے ہیں جس میں User کی درخواست کردہ ٹاپنگز شامل ہیں۔ اس مثال میں 'فرانسیسی فرائز' کی ایک غیر معمولی درخواست ہے۔ پھر ہم درخواست شدہ ٹاپنگز کی List پر Loop کرتے ہیں۔ Loop کے اندر، ہم چیک کرتے ہیں کہ آیا ہر درخواست شدہ ٹاپنگ حقیقتاً دستیاب ٹاپنگز کی List میں ہے یا نہیں۔ اگر یہ دستیاب ہے تو ہم اسے پیزا میں شامل کر لیتے ہیں۔ اگر درخواست شدہ ٹاپنگ List میں دستیاب نہیں ہے، تو else بلاک چلتا ہے جو User کو بتاتا ہے کہ وہ ٹاپنگ دستیاب نہیں ہے۔

یہ code کی ترکیب صاف، معلوماتی output پیدا کرتی ہے:

```
Adding mushrooms.
Sorry, we don't have french fries.
Adding extra cheese.
Finished making your pizza!
```

13.2 Excercise

ایک List بنائیں جس میں پانچ یا زیادہ User نیم ہوں، جس میں 'admin' کا نام بھی شامل ہو۔ تصور کریں کہ آپ کو ایسا code لکھنا ہے جو ہر User کے ویب سائٹ پر لاگ ان ہونے کے بعد انہیں ایک پیغام دے۔ List کو Loop کریں اور ہر User کو ایک پیغام پرنٹ کریں۔

- اگر User کا نام 'admin' ہو، تو ایک خاص پیغام پرنٹ کریں جیسے Hello admin, would you like to see a status report?
- ورنہ، ایک عمومی پیغام پرنٹ کریں جیسے Hello Jaden, thank you for logging in again.

کوئی User نہیں

hello_admin.py میں ایک IF ٹیسٹ شامل کریں تاکہ یہ یقینی بنایا جاسکے کہ User کی List خالی نہیں ہے۔

- اگر List خالی ہو، تو پیغام پرنٹ کریں We-need-to-find-some-users!
- اپنے تمام User نیم List سے نکالیں اور یہ چیک کریں کہ صحیح پیغام پرنٹ ہو رہا ہے۔

User نیم کی جانچ

مندرجہ ذیل کام کریں تاکہ ایک پروگرام بنایا جاسکے جو اس بات کی نقل کرے کہ ویب سائٹس کس طرح یہ یقینی بناتی ہیں کہ ہر کسی کا User نیم منفرد ہو۔

Dr. Muhammad Siddique 03006329919

- پانچ یا زیادہ User نیموں کی List بنائیں جسے current_users کہا جائے۔
- پانچ نئے User نیموں کی ایک اور List بنائیں جسے new_users کہا جائے۔ یہ یقین دہانی کریں کہ ایک یا دو نئے User نیم current_users کی List میں بھی شامل ہوں۔
- new_users کی List کو Loop کریں تاکہ یہ چیک کیا جاسکے کہ ہر نیا User نیم پہلے ہی استعمال ہو چکا ہے یا نہیں۔ اگر وہ پہلے سے استعمال ہو چکا ہو، تو پیغام پرنٹ کریں کہ اس person کو نیا User نیم درج کرنا ہوگا۔ اگر User نیم استعمال نہیں ہوا ہو، تو پیغام پرنٹ کریں کہ User نیم دستیاب ہے۔
- اس بات کو یقینی بنائیں کہ آپ کا موازنہ کیس حساس نہ ہو۔ اگر 'John' پہلے سے استعمال ہو چکا ہو، تو 'JOHN' کو قبول نہیں کیا جائے گا۔ (اس کے لئے آپ کو current_users کی ایک نقل بنانی ہوگی جس میں تمام موجودہ User نیموں کے نچلے کیس ورژن ہوں۔)

Count نمبر

عدد کی ترتیب ان کے List میں مقام کو ظاہر کرتی ہے، جیسے 1st یا 2nd۔ زیادہ تر عددی نمبروں کا اختتام th پر ہوتا ہے، سوائے 1، 2، اور 3 کے۔

- نمبر 1 سے 9 تک ایک List میں محفوظ کریں۔

- List کو Loop کریں۔

- Loop کے اندر fi-elfi-else چین کا استعمال کریں تاکہ ہر نمبر کے لیے صحیح عددی اختتام پرنٹ کیا جا سکے۔ آپ کا output اس طرح ہونا چاہیے: 1st"2nd3rd4th5th6th7th8th"، 9th اور ہر نتیجہ ایک علیحدہ لائن پر ہونا چاہیے۔

14.2 statement-IF کو اسٹائل کریں

اس باب میں ہر مثال میں آپ نے اچھے اسٹائلنگ کے عادات دیکھی ہیں۔ PEP-8 جو مشورہ دیتا ہے وہ یہ ہے کہ conditional-ٹیسٹس کے لیے موازنہ آپریٹرز جیسے ==، <=، اور >= کے ارد گرد ایک جگہ کا استعمال کریں۔ مثال کے طور پر:

```
if age < 4:
```

یہ بہتر ہے بجائے:

```
if age<4:
```

Dr. Muhammad Siddique 03006329919

ایسی جگہ بندی Python کے ذریعے آپ کے code کے سمجھنے کے طریقے پر اثر انداز نہیں ہوتی؛ یہ بس آپ کے اور دوسروں کے لیے آپ کے code کو پڑھنا آسان بناتی ہے۔

Excercise 15.2

5-12. statement-IF کو اسٹائل کرنا

اس باب میں آپ نے جو پروگرامز لکھے ہیں، ان کا جائزہ لیں اور یہ یقینی بنائیں کہ آپ نے اپنے conditional-ٹیسٹس کو مناسب طریقے سے اسٹائل کیا ہے۔

5-13. آپ کے خیالات

اس وقت، آپ ایک بہتر پروگرامر ہیں بنسبت جب آپ نے یہ کتاب شروع کی تھی۔ اب چونکہ آپ کو یہ بہتر سمجھ آگئی ہے کہ حقیقی دنیا کی صورتوں کو پروگرامز میں کس طرح ماڈل کیا جاتا ہے، آپ سوچ سکتے ہیں کہ آپ کچھ مسائل حل کرنے کے بارے میں سوچ رہے ہوں گے جو آپ اپنے پروگرامز کے ذریعے حل کر سکتے ہیں۔ آپ اپنے نئے خیالات نوٹ کریں جو آپ کے ذہن میں آرہے ہیں، جیسے کھیل لکھنا، ڈیٹا سیٹس کا جائزہ لینا، یا ویب ایپلیکیشنز تخلیق کرنا۔

16.2 خلاصہ

اس باب میں آپ نے conditional- ٹیسٹس لکھنا سیکھا جو ہمیشہ True یا False میں تبدیل ہوتے ہیں۔ آپ نے سادہ IF، ات statement- fi-else چیز اور fi-elfi-else چیز لکھنا سیکھا۔ آپ نے ان ڈھانچوں کا استعمال کرتے ہوئے مخصوص شرائط کی نشاندہی کرنا شروع کی جو آپ کو ٹیسٹ کرنی تھیں اور یہ جاننا شروع کیا کہ وہ شرائط آپ کے پروگرامز میں کب پوری ہوتی ہیں۔ آپ نے سیکھا کہ کس طرح کسی List کے مخصوص elements کو دوسرے تمام elements سے مختلف طریقے سے ہینڈل کیا جائے اور ساتھ ہی Loopfor کی کارکردگی کا فائدہ اٹھایا۔ آپ نے Python کے اسٹائل کے مشوروں پر دوبارہ غور کیا تاکہ یہ یقینی بنایا جاسکے کہ آپ کے بڑھتے ہوئے پیچیدہ پروگرامز اب بھی نسبتاً آسانی سے پڑھنے اور سمجھنے کے قابل ہیں۔

باب 6 میں آپ Python کے dictionary- کے بارے میں سیکھیں گے۔ ایک Dictionary ایک List کی طرح ہوتی ہے، لیکن یہ آپ کو معلومات کے ٹکڑوں کو آپس میں جوڑنے کی اجازت دیتی ہے۔ آپ سیکھیں گے کہ dictionary- کو کیسے بنایا جائے، ان کے ذریعے Loop کیسے کیا جائے، اور انہیں وں List اور IF ات statement- کے ساتھ مل کر کیسے استعمال کیا جائے۔ dictionary- کے بارے میں سیکھنا آپ کو حقیقی دنیا کی حالات کی ایک اور وسیع اقسام کو ماڈل کرنے کے قابل بنائے گا۔

Dr. Muhammad Siddique 03006329919

(Dictionaries)dictionary-

اس باب میں آپ سیکھیں گے کہ Python کی dictionary- کو کیسے استعمال کیا جائے، جو آپ کو متعلقہ معلومات کے ٹکڑوں کو جوڑنے کی اجازت دیتی ہیں۔ آپ سیکھیں گے کہ معلومات کو Dictionary میں رکھنے کے بعد اس تک کس طرح access حاصل کی جائے اور اس معلومات کو کس طرح تبدیل کیا جائے۔ چونکہ dictionary- تقریباً لا متناہی مقدار میں معلومات store کر سکتی ہیں، میں آپ کو یہ دکھاؤں گا کہ Dictionary میں موجود ڈیٹا کے ذریعے Loop کیسے چلایا جائے۔ اس کے علاوہ، آپ سیکھیں گے کہ dictionary- کو وں List کے اندر، وں List کو dictionary- کے اندر، اور یہاں تک کہ dictionary- کو دوسرے dictionary- کے اندر کیسے رکھیں۔ dictionary- کو سمجھنا آپ کو حقیقی دنیا کی مختلف elements کو زیادہ درست طریقے سے ماڈل کرنے کی اجازت دیتا ہے۔ آپ ایک Dictionary بنا سکیں گے جو کسی person کی نمائندگی کرے، اور پھر آپ اس person کے بارے میں جتنی معلومات چاہیں store کر سکتے ہیں۔ آپ ان کا نام، عمر، مقام، پیشہ، اور کسی بھی دوسرے پہلو کو جو آپ کسی person کے بارے میں statement- کر سکتے ہیں، store کر سکتے ہیں۔ آپ ایسی معلومات store کر سکتے ہیں جو ایک دوسرے سے میل کھاتی ہو، جیسے کہ word- اور ان کے معنی، لوگوں کے نام اور ان کے پسندیدہ نمبر، پہاڑوں کی List اور ان کی اونچائیاں، وغیرہ۔

Dr. Muhammad Siddique 03006329919

ایک سادہ Dictionary

تصور کریں کہ ایک کھیل ہے جس میں alien (aliens) مختلف رنگوں اور پوائنٹس کی value- وں کے ساتھ ہو سکتے ہیں۔ یہ سادہ Dictionary ایک خاص alien کے بارے میں معلومات store کرتی ہے:

```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0['color'])
print(alien_0['points'])
```

Dictionary: alien_0– alien کے color اور points کی value- کو store کرتی ہے۔ آخری دو لائنیں اس معلومات تک access حاصل کرتی ہیں اور اسے print کرتی ہیں:

```
green
5
```

جیسے کہ زیادہ تر نئے پروگرامنگ تصورات کے ساتھ، dictionary- کا استعمال excercise کی ضرورت ہوتی ہے۔ ایک بار جب آپ dictionary- کے ساتھ تھوڑا کام کریں گے، تو آپ دیکھیں گے کہ وہ کس طرح مؤثر طریقے سے حقیقی دنیا کی صورتوں کو ماڈل کر سکتے ہیں۔

dictionary- کے ساتھ کام کرنا

Python میں Dictionary ایک key-value- جوڑے (key-value pairs) کا مجموعہ ہوتی ہے۔ ہر key ایک value- سے جڑی ہوتی ہے، اور آپ اس key کو استعمال کر کے اس کے ساتھ جڑی value- تک access حاصل کر سکتے ہیں۔ ایک key کی value- ایک نمبر، ایک سٹرنگ، ایک List، یا یہاں تک کہ دوسری Dictionary

بھی ہو سکتی ہے۔ حقیقت میں، آپ Python میں جس بھی شے کو تخلیق کر سکتے ہیں، اسے Dictionary میں value-
کے طور پر استعمال کر سکتے ہیں۔
Python میں، Dictionary بریکٹ () میں بند ہوتی ہے، اور اس میں key-value- جوڑے ہوتے ہیں،
جیسا کہ پہلے کی مثال میں دکھایا گیا:

```
alien_0 = {'color': 'green', 'points': 5}
```

key-value- جوڑا وہ values ہیں جو آپس میں جڑی ہوتی ہیں۔ جب آپ ایک key فراہم کرتے ہیں، Python
اس key سے جڑی value- واپس کرتا ہے۔ ہر key اپنے value- سے کولن (:) کے ذریعے جڑی ہوتی ہے، اور انفرادی
key-value- جوڑے کو مہ (,) سے الگ ہوتے ہیں۔ آپ جتنے چاہیں key-value- جوڑے Dictionary
میں store کر سکتے ہیں۔ سب سے سادہ Dictionary: میں بالکل ایک key-value- جوڑا ہوتا ہے، جیسا کہ اس
ترمیم شدہ alien_0 Dictionary میں دکھایا گیا:

```
alien_0 = {'color': 'green'}
```

یہ alien_0:Dictionary کے بارے میں ایک معلومات store کرتی ہے: alien کا رنگ۔ سٹرنگ 'color'
اس Dictionary میں ایک key ہے اور اس کی جڑی value- 'green' ہے۔

Dr. Muhammad Siddique 03006329919

Dictionary میں value- تک access حاصل کرنا

value- تک access حاصل کرنے کے لیے، Dictionary کا نام فراہم کریں اور پھر key کو square بریکٹس
کے اندر رکھیں، جیسا کہ یہاں دکھایا گیا:

```
alien_0 = {'color': 'green'}
print(alien_0['color'])
```

یہ alien_0:Dictionary میں 'color' key سے جڑی value- واپس کرتا ہے:

```
green
```

آپ Dictionary- میں ایک infinite تعداد میں key-value- جوڑے رکھ سکتے ہیں۔ مثال کے طور پر، یہاں
alien_0 Dictionary- کی اصل شکل ہے جس میں دو key-value- جوڑے ہیں:

```
alien_0 = {'color': 'green', 'points': 5}
```

اب آپ alien_0 کے رنگ یا پوائنٹ value- تک access حاصل کر سکتے ہیں۔ اگر کوئی کھلاڑی اس alien کو
گرا دیتا ہے، تو آپ اس بات کو دیکھ سکتے ہیں کہ کھلاڑی کو کتنے پوائنٹس ملنے چاہئیں، جیسا کہ اس code میں دکھایا گیا:

```
alien_0 = {'color': 'green', 'points': 5}
new_points = alien_0['points']
print(f"You just earned {new_points} points!")
```

```
You just earned 5 points!
```

pairs-Key-Value کو حذف کرنا

جب آپ کو کسی معلومات کی ضرورت نہ ہو جو کہ ڈکشنری میں محفوظ ہو، تو آپ statement-del کا استعمال کر کے کسی Value-Key جوڑی کو مکمل طور پر حذف کر سکتے ہیں۔ del کو صرف ڈکشنری کا نام اور وہ کی چاہیے جسے آپ حذف کرنا چاہتے ہیں۔ مثال کے طور پر، آیئے ہم alien_0 ڈکشنری سے 'points' کی کو حذف کرتے ہیں، ساتھ ہی اس کی value-بھی:

```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)
del alien_0['points']
print(alien_0)
```

statement-del alien_0 ڈکشنری سے 'points' کی کو حذف کرنے کے لئے کہتا ہے اور اس کی value-کو بھی ہٹا دیتا ہے۔ output میں دکھایا گیا ہے کہ 'points' key اور اس کی value-5 حذف ہو گئی ہے، لیکن باقی ڈکشنری متاثر نہیں ہوئی:

```
{'color': 'green', 'points': 5}
{'color': 'green'}
```

نوٹ: یہ بات یاد رکھیں کہ حذف شدہ Value-Key جوڑی کو مستقل طور پر ہٹا دیا جاتا ہے۔

Dr. Muhammad Siddique 03006329919

مشابہ elements کی ڈکشنری

پچھلے مثال میں ایک ہی چیز کے بارے میں مختلف قسم کی معلومات store کی گئی تھی، یعنی ایک گیم میں alien کے بارے میں۔ آپ ایک ڈکشنری کا استعمال کئی elements کے بارے میں ایک ہی قسم کی معلومات store کرنے کے لیے بھی کر سکتے ہیں۔ مثلاً، اگر آپ کئی لوگوں سے ان کی پسندیدہ پروگرامنگ زبان کے بارے میں سوال کریں تو ایک ڈکشنری اس سادہ پول کے نتائج store کرنے کے لیے مفید ہو سکتی ہے:

```
favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'rust',
    'phil': 'python',
}
```

جیسا کہ آپ دیکھ سکتے ہیں، ہم نے ایک بڑی ڈکشنری کو کئی لائنوں میں تقسیم کیا ہے۔ ہر key ایک person کا نام ہے جس نے پول کا جواب دیا اور ہر value- ان کی زبان کی پسند ہے۔ جب آپ کو ایک ڈکشنری کو declare کرنے کے لیے ایک سے زیادہ لائنوں کی ضرورت ہو، تو کھلی آہنی قوس کے بعد انٹر دبائیں۔ پھر اگلی لائن کو ایک سطح (چار اسپیس) انڈنٹ کریں اور پہلا Value-Key جوڑا لکھیں، اس کے بعد کاما لگائیں۔ اس کے بعد جب آپ انٹر دبائیں گے، آپ کا ایڈیٹر خود بخود تمام اگلے pairs-Value-Key کو پہلی جوڑی کے مطابق انڈنٹ کرے گا۔

ایک بار جب آپ ڈکشنری کی declare مکمل کر لیں، تو آخری Value-Key جوڑے کے بعد ایک نیا لائن پر بند آہنی قوس کا اضافہ کریں اور اسے ایک سطح انڈنٹ کریں تاکہ یہ ڈکشنری میں موجود کیز کے ساتھ سیدھ میں آ جائے۔ یہ اچھا عمل ہے کہ آخری Value-Key جوڑے کے بعد بھی کاما شامل کریں تاکہ آپ اگلے لائن پر نیا Value-Key جوڑا شامل کرنے کے لیے تیار ہوں۔

get() کا استعمال

ڈکشنری سے آپ جس value- میں دلچسپی رکھتے ہیں اسے square بریکٹس میں کی کی مدد سے حاصل کرنے کے دوران ایک ممکنہ مسئلہ پیش آ سکتا ہے: اگر آپ جس کی کی درخواست کر رہے ہیں وہ موجود نہ ہو، تو آپ کو ایک ایرر ملے گا۔ آیئے دیکھتے ہیں کہ کیا ہوتا ہے جب آپ ایک alien کے پوائنٹ کی value- طلب کرتے ہیں جس میں پوائنٹ value- سیٹ نہ ہو:

```
alien_0 = {'color': 'green', 'speed': 'slow'}
print(alien_0['points'])
```

یہ ایک ٹریس بیک پیدا کرتا ہے، جو KeyError دکھاتا ہے:

```
KeyError: 'points'
```

Dr. Muhammad Siddique 03006329919

آپ اس ایرر کو ہینڈل کرنا سیکھیں گے، لیکن خاص طور پر ڈکشنری کے لیے، آپ ()method-get کا استعمال کر سکتے ہیں تاکہ ایک ڈیفالٹ value- سیٹ کی جا سکے جو کہ تب واپس آ جائے جب درخواست کردہ کی موجود نہ ہو۔

```
alien_0 = {'color': 'green', 'speed': 'slow'}
point_value = alien_0.get('points', 'No point value assigned.')
print(point_value)
```

اگر کی 'points' ڈکشنری میں موجود ہو، تو آپ کو اس کی value- مل جائے گی۔ اگر یہ موجود نہ ہو، تو آپ کو ڈیفالٹ value- ملے گی۔ اس کیس میں، پوائنٹس موجود نہیں ہیں، اور ہمیں ایک صاف پیغام ملتا ہے بجائے ایک ایرر کے:

```
No point value assigned.
```

اگر یہ امکان ہو کہ آپ جس کی کی درخواست کر رہے ہیں وہ موجود نہ ہو، تو آپ کو square بریکٹس کی بجائے ()get method- استعمال کرنا چاہیے۔

Excercise

person: ایک ڈکشنری کا استعمال کریں تاکہ آپ جس person کو جانتے ہیں، اس کی معلومات store کی جا سکے۔ ان کا پہلا نام، آخری نام، عمر، اور وہ شہر جس میں وہ رہتے ہیں، store کریں۔ آپ کو کیز جیسے first_name، last_name، age، اور city ہونی چاہیے۔ اپنی ڈکشنری میں محفوظ ہر معلومات کو پرنٹ کریں۔

پسندیدہ نمبرز: ایک ڈکشنری کا استعمال کریں تاکہ لوگوں کے پسندیدہ نمبروں کو store کیا جا سکے۔ پانچ ناموں کے بارے میں سوچیں، اور انہیں اپنی ڈکشنری میں کیز کے طور پر استعمال کریں۔ ہر person کے لئے ایک پسندیدہ نمبر سوچیں، اور اسے اپنی ڈکشنری میں value- کے طور پر محفوظ کریں۔ ہر person کا نام اور ان کا پسندیدہ نمبر پرنٹ کریں۔ مزید مزے کے لئے، چند دوستوں سے پول کریں اور اپنے پروگرام کے لئے کچھ اصل ڈیٹا حاصل کریں۔

گلاسری: ایک Python ڈکشنری کو ایک اصلی ڈکشنری کی نقل کرنے کے لئے استعمال کیا جا سکتا ہے۔ تاہم، الجھن سے بچنے کے لیے، ہم اسے گلاسری کہیں گے۔

- پچھلے ابواب میں سیکھے گئے پانچ پروگرامنگ -word- کے بارے میں سوچیں۔ ان -word- کو گلاسری میں کیز کے طور پر استعمال کریں، اور ان کے معانی کو values کے طور پر محفوظ کریں۔

- ہر -word- اور اس کا مطلب خوبصورتی سے فارمیٹ کر کے پرنٹ کریں۔ آپ -word- کو کالن کے ساتھ پرنٹ کر سکتے ہیں اور پھر اس کا مطلب لکھ سکتے ہیں، یا -word- کو ایک لائن پر پرنٹ کر سکتے ہیں اور اس کا مطلب دوسری لائن پر انڈنٹ کر کے پرنٹ کر سکتے ہیں۔ ہر -word-- مفہوم جوڑے کے درمیان ایک خالی لائن ڈالنے کے لئے نیو لائن کریکٹر (\n) کا استعمال کریں۔

Dr. Muhammad Siddique 03006329913

ڈکشنری کے ذریعے Looping

ایک واحد Python ڈکشنری میں صرف چند Value-Key جوڑے یا لاکھوں جوڑے شامل ہو سکتے ہیں۔ چونکہ ڈکشنری بڑی مقدار میں ڈیٹا store کر سکتی ہے، Python آپ کو ڈکشنری کے ذریعے Loop کرنے کی اجازت دیتا ہے۔ ڈکشنریوں کا استعمال مختلف طریقوں سے معلومات store کرنے کے لیے کیا جا سکتا ہے؛ لہذا، انہیں Loop کرنے کے کئی مختلف طریقے موجود ہیں۔ آپ ڈکشنری کے تمام Value-Key pairs کے ذریعے، اس کی کیز کے ذریعے، یا اس کی values کے ذریعے Loop کر سکتے ہیں۔

تمام Value-Key pairs کے ذریعے Looping

مختلف طریقوں سے Looping کو explore کرنے سے پہلے، ہم ایک نئی ڈکشنری پر غور کرتے ہیں جس کا مقصد ویب سائٹ پر ایک User کے بارے میں معلومات store کرنا ہے۔ نیچے دی گئی ڈکشنری ایک شخص کا User نیم، پہلا نام، اور آخری نام store کرے گی:

```
user_0 = {
'username': 'efermi',
'first': 'enrico',
'last': 'fermi',
}
```

آپ user_0 کے بارے میں کوئی بھی معلومات اس باب میں سیکھے گئے طریقوں کے مطابق حاصل کر سکتے ہیں۔ لیکن اگر آپ اس User کی ڈکشنری میں محفوظ تمام معلومات دیکھنا چاہتے ہیں، تو یہ کام آپ loop-for کے ذریعے انجام دے سکتے ہیں:

```
user_0 = {
'username': 'efermi',
'first': 'enrico',
'last': 'fermi',
}

for key, value in user_0.items():
    print(f"\nKey: {key}")
    print(f"Value: {value}")
```

ڈکشنری کے لیے Loop-for لکھنے کے لیے، آپ ان دو variable کے لیے نام تخلیق کرتے ہیں جو ہر Key-Value جوڑے میں key اور value-کو پکڑیں گے۔ آپ ان variable کے لیے جو چاہیں نام منتخب کر سکتے ہیں۔ یہ code اسی طرح کام کرے گا اگر آپ variable کے ناموں کے لیے اختصارات استعمال کریں جیسے:

```
for k, v in user_0.items():
```

Dr. Muhammad Siddique 03006229919

for-statement کے دوسرے حصے میں ڈکشنری کا نام آتا ہے، جس کے بعد ()items method آتا ہے جو Key-Value pairs کا ایک سلسلہ واپس کرتا ہے۔ for-Loop پھر ان pairs میں سے ہر ایک کو فراہم کردہ variable میں assign کرتا ہے۔ اس مثال میں، ہم variable کا استعمال کرتے ہوئے ہر کی کے ساتھ اس سے متعلقہ value-کو پرنٹ کرتے ہیں۔ پہلی ()print کال میں ""اس بات کو یقینی بناتا ہے کہ ہر Key-Value جوڑے کے درمیان ایک خالی لائن پرنٹ ہو۔

مثال:

```
Key: username
Value: efermi
Key: first
Value: enrico
Key: last
Value: fermi
```

تمام Key-Value pairs کے ذریعے Looping خاص طور پر ایسی ڈکشنریوں کے لیے مفید ہوتی ہے جیسے کہ favorite_languages.py کی مثال، جو مختلف کیز کے لیے اسی طرح کی معلومات store کرتی ہے۔ اگر آپ favorite_languages ڈکشنری کے ذریعے Loop کرتے ہیں، تو آپ ہر شخص کا نام اور ان کی پسندیدہ پروگرامنگ زبان حاصل کرتے ہیں۔ چونکہ کیز ہمیشہ کسی شخص کے نام کو ظاہر کرتی ہیں اور value-ہمیشہ ایک زبان ہوتی ہے، ہم Loop میں key اور value-کی بجائے نام اور زبان کے variable استعمال کریں گے۔ اس سے Loop کے اندر جو ہو رہا ہے، اسے سمجھنا آسان ہو جائے گا:

```
favorite_languages = {
'jen': 'python',
'sarah': 'c',
```

```
'edward': 'rust',
'phil': 'python',
}

for name, language in favorite_languages.items():
    print(f"{name.-title()}'s favorite language is {language.-title()}.")
```

یہ Python-code کو ہدایت دیتا ہے کہ وہ ڈکشنری میں ہر Value-Key جوڑے کے ذریعے Loop کرے۔ جیسے جیسے یہ ہر جوڑے کو پروسیس کرتا ہے، کی کو variable-name میں اور value- کو variable-language میں assign کرتا ہے۔ ان وضاحتی ناموں کی مدد سے پرنٹ () کال کیا کر رہی ہے، یہ دیکھنا زیادہ آسان ہوتا ہے۔

output:

```
Jen's favorite language is Python.
Sarah's favorite language is C.
Edward's favorite language is Rust.
Phil's favorite language is Python.
```

یہ قسم کی Looping اتنی ہی اچھے طریقے سے کام کرے گی اگر ہماری ڈکشنری میں ایک ہزار یا یہاں تک کہ لاکھوں لوگوں کے پول کے نتائج store کیے جائیں۔

Dr. Muhammad Siddique 03006329969

ڈکشنری کی تمام کیز کے ذریعے Looping

جب آپ کو ڈکشنری میں موجود تمام values کے ساتھ کام کرنے کی ضرورت نہ ہو، تو keys () method مفید ثابت ہوتی ہے۔ آیئے favorite_languages ڈکشنری کے ذریعے Loop کرتے ہیں اور ان تمام لوگوں کے نام پرنٹ کرتے ہیں جنہوں نے پول میں حصہ لیا:

```
favorite_languages = {
'jen': 'python',
'sarah': 'c',
'edward': 'rust',
'phil': 'python',
}

for name in favorite_languages.keys():
    print(name.-title())
```

یہ Python-for-Loop کو ہدایت دیتا ہے کہ وہ favorite_languages ڈکشنری کی تمام کیز کو نکالے اور انہیں ایک وقت میں variable-name میں assign کرے۔ output ان تمام لوگوں کے نام دکھائے گا جنہوں نے پول میں حصہ لیا:

output:

```
Jen
Sarah
Edward
Phil
```

ڈکشنری کے ذریعے Loop کرتے وقت کیز کے ذریعے Loop کرنا دراصل ڈیفالٹ رویہ ہوتا ہے، لہذا یہ code بالکل ویسا ہی output دے گا اگر آپ نے یہ لکھا ہو:

```
for name in favorite_languages:
```

آپ method-keys () کو واضح طور پر استعمال کر سکتے ہیں اگر یہ آپ کے code کو پڑھنے میں آسانی پیدا کرتا ہے، یا اگر آپ چاہیں تو اسے چھوڑ بھی سکتے ہیں۔

آپ Loop کے اندر کسی بھی مطلوبہ کی کے ساتھ اس سے متعلقہ value- تک access حاصل کر سکتے ہیں۔ آیئے ہم دو دوستوں کو ان کی پسندیدہ زبانوں کے بارے میں پیغام پرنٹ کرتے ہیں۔ ہم پہلے کی طرح ڈکشنری میں موجود ناموں کے ذریعے Loop کریں گے، لیکن جب نام ہمارے کسی دوست سے میل کھائے گا، ہم ان کی پسندیدہ زبان کے بارے میں پیغام دکھائیں گے:

```
favorite_languages = {
--snip--
}

friends = ['phil', 'sarah']

for name in favorite_languages.keys():
    print(f"Hi {name.-title()}.")
    if name in friends:
        language = favorite_languages[name].-title()
        print(f"\t{name.-title()}, I see you love {language}!")
```

Dr. Muhammad Siddique 03006329919

سب کے نام پرنٹ ہو جاتے ہیں، لیکن ہمارے دوستوں کو ایک خاص پیغام ملتا ہے:

output:

```
Hi Jen.
Hi Sarah.
Sarah, I see you love C!
Hi Edward.
Hi Phil.
Phil, I see you love Python!
```

آپ methodkeys () کا استعمال یہ معلوم کرنے کے لیے بھی کر سکتے ہیں کہ کیا کسی خاص شخص نے پول میں حصہ لیا تھا۔ اس بار ہم یہ معلوم کریں گے کہ کیا Erin نے پول میں حصہ لیا تھا:

```
if 'erin' not in favorite_languages.keys():
    print("Erin, please take our poll!")
```

methodkeys () صرف Looping کے لیے نہیں ہے: یہ دراصل تمام کیز کا ایک سلسلہ واپس کرتی ہے، اور IF-statement صرف یہ چیک کرتا ہے کہ کیا 'erin' اس سلسلے میں موجود ہے۔ چونکہ وہ نہیں ہے، ایک پیغام پرنٹ کیا جاتا ہے جو اسے پول میں حصہ لینے کی دعوت دیتا ہے:

output:

```
Erin, please take our poll!
```

17.2 ڈکشنری کی keys مخصوص ترتیب میں Loop کرنا

ڈکشنری کے ذریعے Loop کرنے سے آئٹمز اسی ترتیب میں واپس آتی ہیں جس میں وہ داخل کی گئی تھیں۔ تاہم، بعض اوقات آپ چاہتے ہیں کہ ڈکشنری کو کسی مختلف ترتیب میں Loop کیا جائے۔

ایک طریقہ یہ ہے کہ آپ فور Loop میں keys واپس آنے پر انہیں ترتیب دیں۔ آپ sorted() فنکشن کا استعمال کر کے keys ترتیب میں حاصل کر سکتے ہیں:

```
favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'rust',
    'phil': 'python',
}
for name in sorted(favorite_languages.keys()):
    print(f"{name.-title()}, thank you for taking the poll.")
```

یہ For اسٹیٹمنٹ دوسرے For اسٹیٹمنٹس کی طرح ہے، سوائے اس کے کہ ہم نے sorted() فنکشن کو dictionary.keys() method- کے ارد گرد لپیٹ دیا ہے۔ اس کا مطلب یہ ہے کہ Python ڈکشنری کی تمام keys حاصل کرے گا اور Loop شروع کرنے سے پہلے انہیں ترتیب دے گا۔ نتیجہ میں وہ سب لوگ دکھائے جائیں گے جنہوں نے پول میں حصہ لیا، اور ان کے نام ترتیب میں ظاہر ہوں گے:

Dr. Muhammad Siddique 03006329919

- Edward, thank you for taking the poll.
- Jen, thank you for taking the poll.
- Phil, thank you for taking the poll.
- Sarah, thank you for taking the poll.

18.2 ڈکشنری میں تمام values کے ذریعے Loop کرنا

اگر آپ بنیادی طور پر ان values میں دلچسپی رکھتے ہیں جو ایک ڈکشنری میں موجود ہیں، تو آپ method-values() کا استعمال کر کے values کی ایک سیریز حاصل کر سکتے ہیں بغیر کسی key کے۔ مثال کے طور پر، فرض کریں کہ ہم صرف ان تمام زبانوں کی List چاہتے ہیں جو ہمارے پروگرامنگ زبان کے پول میں منتخب کی گئی ہیں، بغیر اس کے کہ ہر زبان کو منتخب کرنے والے شخص کا نام ہو:

```
favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'rust',
```

```
    'phil': 'python',
}
print("The following languages have been mentioned:")
for language in favorite_languages.values():
    print(language.-title())
```

یہ فور اسٹیٹمنٹ ڈکشنری سے ہر-value نکال کر اسے variable-language میں محفوظ کرتا ہے۔ جب یہ values پرنٹ کی جاتی ہیں تو ہمیں تمام منتخب شدہ زبانوں کی List ملتی ہے:

- mentioned:beenhavelanguagesfollowingThe
- Python
- C
- Rust
- Python

یہ طریقہ ڈکشنری سے تمام values نکالتا ہے بغیر کسی دہرائی کے چیک کیے۔ یہ طریقہ چھوٹی تعداد میں values کے ساتھ اچھا کام کرتا ہے، لیکن اگر پول میں زیادہ تعداد میں جوابات ہوں، تو یہ ایک بہت زیادہ دہرائی جانے والی List پیدا کرے گا۔ ہر زبان کو بغیر تکرار کے دیکھنے کے لیے ہم set استعمال کر سکتے ہیں۔ set ایک مجموعہ ہوتا ہے جس میں ہر آئٹم منفرد ہوتا ہے:

Dr. Muhammad Siddique 03006329019

```
favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'rust',
    'phil': 'python',
}
print("The following languages have been mentioned:")
for language in set(favorite_languages.values()):
    print(language.-title())
```

جب آپ set() کو ایک ایسے مجموعے کے ارد گرد لپیٹتے ہیں جس میں نقل شدہ آئٹمز ہوں، Python ان منفرد آئٹمز کو شناخت کرتا ہے اور ان سے set بناتا ہے۔ یہاں ہم set() استعمال کرتے ہیں تاکہ favorite_languages.values() میں منفرد زبانوں کو نکال سکیں۔

- The-following-languages-have-been-mentioned:
- Python
- C
- Rust

جیسے جیسے آپ Python کے بارے میں مزید سیکھیں گے، آپ زبان کی ایک بلٹ ان خصوصیت کا سامنا کریں گے جو آپ کو اپنے ڈیٹا کے ساتھ بالکل وہی کرنے میں مدد دے گی جو آپ چاہتے ہیں۔
نوٹ: آپ براہ راست سیٹ بنانے کے لیے بریکٹ کا استعمال کر سکتے ہیں اور elements کو کاما سے جدا کر سکتے ہیں:

```
>>> languages = {'python', 'rust', 'python', 'c'}
>>> languages
{'rust', 'python', 'c'}
```

سیٹ اور ڈکشنری کو آسانی سے غلطی سے ملایا جا سکتا ہے کیونکہ دونوں بریکٹس میں لپٹے ہوتے ہیں۔ جب آپ بریکٹس دیکھیں لیکن کوئی key-value جوڑا نہ ہو، تو آپ غالباً سیٹ دیکھ رہے ہوں گے۔

19.2 Nesting

کبھی کبھار آپ کو ایک List میں متعدد ڈکشنریاں store کرنی ہوتی ہیں، یا کسی List کے elements کو ایک ڈکشنری میں قدر کے طور پر استعمال کرنا ہوتا ہے۔ اسے نسٹنگ (Nesting) کہتے ہیں۔ آپ ڈکشنریوں کو List کے اندر، elements کی List کو ڈکشنری کے اندر، یا یہاں تک کہ ایک ڈکشنری کو دوسری ڈکشنری کے اندر بھی رکھ سکتے ہیں۔ نسٹنگ ایک طاقتور خصوصیت ہے، جیسا کہ اگلے مثالوں میں دکھایا جائے گا۔

Dr. Muhammad Siddique 03506325919

1.19.2 ڈکشنریوں کی List

alien_0 ڈکشنری میں ایک ایلین کے بارے میں مختلف معلومات ہیں، لیکن اس کے پاس دوسرے ایلین کے بارے میں معلومات store کرنے کی جگہ نہیں ہے، اور نہ ہی Aliens کا پورا اسکرین۔ آپ Aliens کے بیڑے کا انتظام کس طرح کر سکتے ہیں؟ ایک طریقہ یہ ہے کہ آپ Aliens کی ایک List بنائیں جس میں ہر ایلین کے بارے میں معلومات کے ساتھ ایک ڈکشنری ہو۔ مثلاً، نیچے دی گئی code تین Aliens کی List بناتی ہے:

```
alien_0 = {'color': 'green', 'points': 5}
alien_1 = {'color': 'yellow', 'points': 10}
alien_2 = {'color': 'red', 'points': 15}
aliens = [alien_0, alien_1, alien_2]

for alien in aliens:
    print(alien)
```

سب سے پہلے ہم تین ڈکشنریاں بناتے ہیں، ہر ایک مختلف ایلین کی نمائندگی کرتی ہے۔ پھر ان ڈکشنریوں کو ایک List میں store کرتے ہیں جس کا نام aliens ہے۔ آخر میں ہم اس List کے ذریعے Loop کرتے ہیں اور ہر ایلین کو پرنٹ کرتے ہیں:

```
{'color': 'green', 'points': 5}
{'color': 'yellow', 'points': 10}
{'color': 'red', 'points': 15}
```

ایک زیادہ حقیقت پسندانہ مثال میں تین سے زیادہ ایلین ہوں گے اور code خودکار طور پر ہر ایلین کو پیدا کرے گا۔ نیچے دی گئی مثال میں ہم range() فنکشن کا استعمال کرتے ہوئے Aliens30 کا بیڑہ تیار کرتے ہیں:

```
# Make an empty list for storing aliens.
aliens = []
# Make 30 green aliens.
for alien_number in range(30):
    new_alien = {'color': 'green', 'points': 5, 'speed': 'slow'
   }
    aliens.append(new_alien)
# Show the first 5 aliens.
for alien in aliens[:5]:
    print(alien)
print("...")
# Show how many aliens have been created.
print(f"Total number of aliens: {len(aliens)}")
```

یہ مثال ایک خالی List سے شروع ہوتی ہے تاکہ تمام Aliens کو store کیا جاسکے جو بنائے جائیں گے۔ range() فنکشن ایک سلسلہ فراہم کرتا ہے، جو پائیتھون کو بتاتا ہے کہ ہمیں Loop کتنی بار چلانا ہے۔ ہر بار جب Loop چلتا ہے، ہم ایک نیا ایلین بناتے ہیں اور پھر ہر نئے ایلین کو aliens List میں شامل کرتے ہیں۔ ہم پہلے پانچ Aliens کو پرنٹ کرنے کے لئے ایک slice کا استعمال کرتے ہیں، اور آخر میں ہم List کی لمبائی کو پرنٹ کرتے ہیں تاکہ یہ ثابت کیا جاسکے کہ ہم نے دراصل Aliens30 کا پورا بیڑہ تیار کیا ہے:

Dr. Muhammad Siddique 03009829999

```
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
...
Total number of aliens: 30
```

یہ تمام Aliens کی خصوصیات ایک جیسی ہیں، لیکن پائیتھون انہیں علیحدہ elements کے طور پر سمجھتا ہے، جس سے ہمیں ہر ایلین کو انفرادی طور پر تبدیل کرنے کی اجازت ملتی ہے۔

2.19.2 Aliens کے ساتھ کام کرنا

آپ اس بیڑے کے ساتھ کس طرح کام کریں گے؟ فرض کریں کہ کھیل کا ایک حصہ ہے جس میں ایلین رنگ تبدیل کرتے ہیں اور جیسے جیسے کھیل بڑھتا ہے، وہ تیز رفتاری سے حرکت کرتے ہیں۔ جب رنگ تبدیل کرنے کا وقت آئے، ہم ایک for Loop اور ایک IF-statement کا استعمال کر سکتے ہیں تاکہ Aliens کا رنگ تبدیل کیا جاسکے۔ مثال کے طور پر، اگر ہمیں پہلے تین Aliens کو پیلے رنگ کا بنانا ہو، جو درمیانہ رفتار اور 10 پوائنٹس والے ہوں، تو ہم یہ کر سکتے ہیں:

```
# Make an empty list for storing aliens.
aliens = []
# Make 30 green aliens.
for alien_number in range (30):
```

```
    new_alien = {'color': 'green', 'points': 5, 'speed': 'slow'}
    aliens.append(new_alien)

for alien in aliens[:3]:
    if alien['color'] == 'green':
        alien['color'] = 'yellow'
        alien['speed'] = 'medium'
        alien['points'] = 10

# Show the first 5 aliens.
for alien in aliens[:5]:
    print(alien)
print("...")
```

چونکہ ہم پہلے تین Aliens کو تبدیل کرنا چاہتے ہیں، ہم صرف پہلے تین Aliens کے لئے Loop کرتے ہیں۔ اس وقت تمام ایلین سبز رنگ کے ہیں، لیکن یہ ہمیشہ ایسا نہیں رہے گا، لہذا ہم ایک statement-IF لکھتے ہیں تاکہ ہم صرف سبز Aliens کو ہی تبدیل کریں۔ اگر ایلین سبز ہے تو ہم اس کا رنگ 'پیلا'، رفتار 'درمیانہ' اور پوائنٹس کی value-10 کر دیتے ہیں، جیسا کہ نیچے دکھایا گیا ہے:

```
{'color': 'yellow', 'points': 10, 'speed': 'medium'}
{'color': 'yellow', 'points': 10, 'speed': 'medium'}
{'color': 'yellow', 'points': 10, 'speed': 'medium'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
...
```

Dr. Muhammad Siddique 03006329919

آپ اس Loop کو مزید بڑھا سکتے ہیں اگر آپ ایک el-if بلاک شامل کریں جو پیلے رنگ کے Aliens کو سرخ، تیز حرکت کرنے والے Aliens میں تبدیل کر دے، جن کے ہر ایک کی value-15 پوائنٹس ہو۔ پورے پروگرام کو دوبارہ دکھائے بغیر، وہ Loop اس طرح دکھے گا:

```
for alien in aliens[0:3]:
    if alien['color'] == 'green':
        alien['color'] = 'yellow'
        alien['speed'] = 'medium'
        alien['points'] = 10
    elif alien['color'] == 'yellow':
        alien['color'] = 'red'
        alien['speed'] = 'fast'
        alien['points'] = 15
```

یہ عام ہے کہ آپ ایک List میں متعدد ڈکشنریاں store کرتے ہیں جب کہ ہر ڈکشنری ایک شے کے بارے میں کئی قسم کی معلومات رکھتی ہے۔ مثال کے طور پر، آپ ویب سائٹ پر ہر User کے لئے ایک ڈکشنری بنا سکتے ہیں،، اور ان ڈکشنریوں کو ایک List میں store کر سکتے ہیں جس کا نام users ہو۔ List میں موجود تمام ڈکشنریوں کا ڈھانچہ یکساں ہونا چاہیے تاکہ آپ List کے ذریعے Loop کر سکیں اور ہر ڈکشنری elements کے ساتھ یکساں طور پر کام کر سکیں۔

20.2 ایک Dictionary میں List

کبھی کبھی یہ مفید ہوتا ہے کہ آپ کسی Dictionary کے اندر List رکھیں۔ مثال کے طور پر، یہ دیکھیں کہ آپ کسی پیزا کی وضاحت کیسے کر سکتے ہیں جو کوئی آرڈر کر رہا ہے۔ اگر آپ صرف List کا استعمال کریں، تو آپ جو کچھ بھی store کر سکتے ہیں وہ صرف پیزا کی ٹاپنگز کی List ہو گی۔ لیکن اگر آپ Dictionary استعمال کریں تو ٹاپنگز کی List پیزا کی وضاحت میں صرف ایک پہلو ہو سکتی ہے۔

مندرجہ ذیل مثال میں، ہر پیزا کے لیے دو قسم کی معلومات محفوظ کی گئی ہیں: کرسٹ کی قسم اور ٹاپنگز کی۔ List ٹاپنگز کی 'toppings' List کی key سے منسلک ایک value- ہے۔ List میں موجود elements تک access حاصل کرنے کے لیے، ہم Dictionary کے نام اور 'toppings' کی key دیتے ہیں، جیسے ہم کسی بھی value- میں کرتے ہیں۔ ایک واحد value- کی بجائے، ہم ٹاپنگز کی List حاصل کرتے ہیں:

```
pizza = {
  'crust': 'thick',
  'toppings': ['mushrooms', 'extra cheese'],
}

print(f"{pizza['crust']}-

for topping in pizza['toppings']:
  print(f"\t{topping}")
```

Dr. Muhammad Siddique 03006329919

ہم سب سے پہلے ایک Dictionary بناتے ہیں جو پیزا کے بارے میں معلومات رکھتا ہے جو آرڈر کیا گیا ہے۔ Dictionary میں ایک key 'crust' ہے، اور اس کے ساتھ منسلک value- 'thick' ہے۔ اگلی، key، 'toppings' ایک List کی value- رکھتی ہے جو تمام آرڈر شدہ ٹاپنگز کو store کرتی ہے۔ ہم آرڈر کا خلاصہ پیش کرنے سے پہلے پیزا بناتے ہیں۔ جب آپ کسی طویل لائن کو print() میں توڑنا چاہتے ہیں تو ایک مناسب مقام منتخب کریں جہاں لائن کو توڑا جائے، اور لائن کے آخر میں ایک اقتباس کی علامت ڈالیں۔ اگلی لائن میں انڈینٹ کریں، ایک اقتباس کی علامت لگائیں، اور سلسلے کو جاری رکھیں۔ Python خود بخود تمام سٹرنگز کو جوڑ دے گا جو اس کے اندر پائی جاتی ہیں۔ ٹاپنگز پر پرنٹ کرنے کے لیے ہم ایک for-loop لکھتے ہیں۔ 'toppings' کی key کے ذریعے، Python ٹاپنگز کی List کو Dictionary سے حاصل کرتا ہے۔ اس کے بعد کی output پیزا کا خلاصہ پیش کرتی ہے جو ہم بنانے والے ہیں:

```
آپ
 □نے thick-crust پیزا آرڈر کیا □□مندرجہ
 □ساتھ □ کے □اپنگز ذیل:
mushrooms
extra cheese
```

آپ جب چاہیں Dictionary میں List رکھ سکتے ہیں جب آپ چاہتے ہوں کہ ایک ہی key سے ایک سے زیادہ value- میں منسلک ہوں۔ پہلے والی مثال میں پسندیدہ پروگرامنگ زبانوں کے حوالے سے، اگر ہم ہر شخص کے جوابات کو ایک List میں store کرتے، تو لوگ ایک سے زیادہ پسندیدہ زبانیں منتخب کر سکتے تھے۔ جب ہم Dictionary پر Loop

کرتے، تو ہر شخص سے منسلک value-اب ایک زبان کی بجائے زبانوں کی List ہوتی۔

21.2 Dictionary میں Dictionary

آپ ایک Dictionary کو دوسری Dictionary میں داخل کر سکتے ہیں، لیکن جب آپ یہ کرتے ہیں تو آپ کا code پیچیدہ ہو سکتا ہے۔ مثال کے طور پر، اگر آپ کے پاس ایک ویب سائٹ کے کئی users ہوں، ہر ایک کا منفرد User نیم ہو، تو آپ User نیم کو Dictionary میں key کے طور پر استعمال کر سکتے ہیں۔ پھر آپ ہر User کے بارے میں معلومات store کرنے کے لیے Dictionary کو User نیم کے ساتھ منسلک value- کے طور پر استعمال کر سکتے ہیں۔ درج ذیل List میں، ہم ہر User کے بارے میں تین قسم کی معلومات محفوظ کرتے ہیں: ان کا پہلا نام، آخری نام، اور مقام۔ ہم یہ معلومات User نیم کے ساتھ منسلک Dictionary کو Loop کر کے حاصل کریں گے:

```
# users کے بارے میں معلومات store کریں
users = {
  'aeinstein': {
    'first': 'albert',
    'last': 'einstein',
    'location': 'princeton',
  },
  'mcurie': {
    'first': 'marie',
    'last': 'curie',
    'location': 'paris',
  },
}

for username, user_info in users.items():
  print(f"\nUser کا نام: {username}")
  full_name = f"{user_info['first']} {user_info['last']}"
  location = user_info['location']
  print(f"\مکمل نام: {full_name.-title()}")
  print(f"\مقام: {location.-title()}")
```

Dr. Muhammad Siddique 03006329919

ہم سب سے پہلے users نام کی ایک ڈکشنری ڈی کلیئر کرتے ہیں جس میں دو keys ہیں: aeinstein اور mcurie۔ ہر key کے ساتھ منسلک value ایک ڈکشنری ہے جس میں ہر User کا پہلا نام، آخری نام، اور مقام شامل ہوتا ہے۔ پھر ہم users ڈکشنری پر loop کرتے ہیں۔ Python ہر key کو username ویری ایبل میں محفوظ کرتا ہے، اور ہر User سے منسلک ڈکشنری کو user_info ویری ایبل میں رکھتا ہے۔ جب ہم مرکزی ڈکشنری پر loop کر رہے ہوتے ہیں، تو ہم User کا یوزر نیم پرنٹ کرتے ہیں۔ پھر ہم اندرونی ڈکشنری تک رسائی حاصل کرتے ہیں۔ user_info ویری ایبل، جس میں User کی معلومات موجود ہوتی ہیں، تین keys پر مشتمل ہوتی ہے: first, last، اور location۔ ہم ہر key کا استعمال کرتے ہوئے ہر User کا مکمل نام اور مقام تیار کرتے ہیں، اور پھر ان کے بارے میں ایک مختصر خلاصہ پرنٹ کرتے ہیں:

```
کا
```

```
User نام: aeinsteinمکمل
 نام: Albert Einsteinمقام
: Princetonکا

User نام: mcurieمکمل
 نام: Marie Curieمقام
: Paris
```

نوٹ کریں کہ ہر User کی Dictionary کا ڈھانچہ ایک جیسا ہے۔ اگرچہ Python میں یہ ضروری نہیں ہے، لیکن یہ ڈھانچہ اندرونی dictionary- کو زیادہ آسانی سے کام کرنے کے قابل بناتا ہے۔ اگر ہر User کی Dictionary میں مختلف keys ہوتیں، تو Loop کے اندر کا code مزید پیچیدہ ہو جاتا۔

Dr. Muhammad Siddique 03006329919

22.2 inputUser اور Loop-While

اکثر پروگرامز کو اینڈ User کے مسئلے کو حل کرنے کے لیے لکھا جاتا ہے۔ ایسا کرنے کے لیے، آپ کو عموماً User سے کچھ معلومات حاصل کرنے کی ضرورت ہوتی ہے۔ مثلاً، فرض کریں کہ کوئی شخص یہ جاننا چاہتا ہے کہ آیا وہ ووٹ دینے کے لیے عمر کے لحاظ سے کافی ہے یا نہیں۔ اگر آپ ایک پروگرام لکھتے ہیں جو اس سوال کا جواب دے، تو آپ کو User کی عمر معلوم کرنا ہوگی تاکہ آپ جواب فراہم کر سکیں۔ پروگرام کو User سے ان کی عمر درج کرنے کے لیے کہنا پڑے گا؛ جب پروگرام کو یہ input مل جائے گا، تو یہ اسے ووٹنگ کی عمر کے ساتھ موازنہ کرے گا تاکہ یہ پتہ چل سکے کہ User عمر کے لحاظ سے کافی ہے یا نہیں اور پھر نتیجہ رپورٹ کرے گا۔

اس باب میں آپ سیکھیں گے کہ inputUser کو کیسے قبول کریں تاکہ آپ کا پروگرام اس کے ساتھ کام کر سکے۔ جب آپ کے پروگرام کو کسی نام کی ضرورت ہوگی، تو آپ User سے نام پوچھنے کے قابل ہوں گے۔ جب آپ کے پروگرام کو ناموں کی فہرست کی ضرورت ہوگی، تو آپ User سے ناموں کی سیریز پوچھنے کے قابل ہوں گے۔ ایسا کرنے کے لیے، آپ input () فنکشن کا استعمال کریں گے۔

آپ یہ بھی سیکھیں گے کہ پروگرامز کو اس وقت تک کیسے چلایا جائے جب تک User چاہے، تاکہ وہ جتنی بھی معلومات چاہیں درج کر سکیں؛ پھر آپ کا پروگرام ان معلومات کے ساتھ کام کر سکے گا۔ آپ پائتھون کے Loop-while کا استعمال کریں گے تاکہ پروگرام اس وقت تک چلتا رہے جب تک مخصوص شرائط سچی رہیں۔

inputUser کے ساتھ کام کرنے کی صلاحیت اور پروگرام کے چلنے کے دوران اپنے کو کنٹرول کرنے کی صلاحیت کے ساتھ، آپ مکمل طور پر انٹرایکٹو پروگرامز لکھنے کے قابل ہوں گے۔

Dr. Muhammad Siddique 03006329919

23.2 input() فنکشن کیسے کام کرتا ہے

input () فنکشن آپ کے پروگرام کو روکتا ہے اور User سے کچھ متن درج کرنے کا انتظار کرتا ہے۔ جب پائتھون User کا input وصول کر لیتا ہے، تو وہ اس input کو ایک variable میں محفوظ کر لیتا ہے تاکہ آپ اس کے ساتھ آسانی سے کام کر سکیں۔

مثال کے طور پر، درج ذیل پروگرام User سے کچھ متن درج کرنے کے لیے کہتا ہے، پھر اس پیغام کو دوبارہ User کو دکھاتا ہے:

```
parrot.py
message = input("Tell me something, and I will repeat it back
    to you: ")
print(message)
```

input () فنکشن ایک دلیل لیتا ہے: وہ پرامپٹ جو ہم User کو دکھانا چاہتے ہیں تاکہ وہ جان سکے کہ انہیں کس قسم کی معلومات درج کرنی ہیں۔ اس مثال میں، جب پائتھون پہلی لائن چلائے گا، User کو پرامپٹ "Tell me something, and I will repeat it back to you: " نظر آئے گا۔ پروگرام اس وقت تک رکے گا جب تک

User اپنا جواب درج کر کے ENTER نہیں دباتا۔ جواب message-variable میں محفوظ ہو جاتا ہے، پھر print(message) input کو دوبارہ User کو دکھاتا ہے:

```
Tell me something, and I will repeat it back to you: Hello
    everyone!
Hello everyone!
```

نوٹ: بعض ٹیکسٹ ایڈیٹرز وہ پروگرامز نہیں چلاتے جو User سے input مانگتے ہیں۔ آپ ان ایڈیٹرز کو input کے لیے پروگرام لکھنے کے لیے استعمال کر سکتے ہیں، لیکن آپ کو ان پروگرامز کو ٹرمینل سے چلانے کی ضرورت ہو گی۔

24.2 واضح پرامپٹس لکھنا

جب بھی آپ input() فنکشن استعمال کرتے ہیں، تو آپ کو ایک واضح، آسان پرامپٹ شامل کرنا چاہیے جو User کو بالکل بتائے کہ آپ کس قسم کی معلومات چاہ رہے ہیں۔ ایسا کوئی بھی statement جو User کو بتائے کہ انہیں کیا داخل کرنا چاہیے، کام کرے گا۔ مثلاً:

```
greeter.py
name = input("Please enter your name: ")
print(f"\nHello, {name}!")
```

Dr. Muhammad Siddique 03006329919

اپنے پرامپٹس کے آخر میں ایک اسپیس شامل کریں (پچھلی مثال میں کالن کے بعد) تاکہ پرامپٹ اور User کے جواب کے درمیان فرق واضح ہو سکے اور User کو یہ سمجھنے میں مدد ملے کہ انہیں کہاں اپنا متن درج کرنا ہے۔

```
Please enter your name: Eric
Hello, Eric!
```

کبھی کبھار آپ چاہتے ہیں کہ پرامپٹ ایک سے زیادہ لائنوں پر مشتمل ہو۔ مثلاً، آپ User کو یہ بتانا چاہتے ہیں کہ آپ کچھ مخصوص input کیوں مانگ رہے ہیں۔ آپ اپنا پرامپٹ ایک variable میں محفوظ کر سکتے ہیں اور اس variable کو input() فنکشن میں پاس کر سکتے ہیں۔ اس سے آپ کو اپنے پرامپٹ کو کئی لائنوں میں تعمیر کرنے کی سہولت ملتی ہے، پھر ایک صاف ستھری input() اسٹیٹمنٹ لکھ سکتے ہیں۔

```
greeter.py
prompt = "If you share your name, we can personalize the
    messages you see."
prompt += "\nWhat is your first name? "
name = input(prompt)
print(f"\nHello, {name}!")
```

یہ مثال ایک طریقہ دکھاتی ہے جس سے آپ ملٹی لائن سٹرنگ بنا سکتے ہیں۔ پہلی لائن پیغام کے پہلے حصے کو variable prompt میں محفوظ کرتی ہے۔ دوسری لائن میں آپریٹر += وہ سٹرنگ جو prompt میں محفوظ تھی، اس کے آخر میں نیا سٹرنگ جوڑ دیتا ہے۔

prompt اب دو لائنوں پر پھیل چکا ہے، پھر سوالیہ نشان کے بعد اسپیس بھی شامل ہے تاکہ یہ واضح ہو سکے:

```
If you share your name, we can personalize the messages you see
    .
What is your first name? Eric
Hello, Eric!
```

25.2 int() کا استعمال کر کے عددی input حاصل کرنا

جب آپ input () فنکشن استعمال کرتے ہیں، تو پائتھون User کی طرف سے درج کی جانے والی ہر چیز کو ایک سٹرنگ کے طور پر interpret کرتا ہے۔ درج ذیل انٹرپرٹر سیشن کو دیکھیں، جو User کی عمر مانگتا ہے:

```
>>> age = input("How old are you? ")
How old are you? 21
>>> age
'21'
```

User عدد 21 درج کرتا ہے، لیکن جب ہم پائتھون سے age کی value- پوچھتے ہیں، تو یہ '21' واپس کرتا ہے، جو کہ عددی value- کا سٹرنگ نمائندگی ہے۔

ہم جانتے ہیں کہ پائتھون نے input کو ایک سٹرنگ کے طور پر interpret کیا کیونکہ عدد اب کوٹیشن مارک کے اندر ہے۔ اگر آپ صرف input کو پرنٹ کرنا چاہتے ہیں تو یہ ٹھیک ہے، لیکن اگر آپ input کو ایک عددی value- کے طور پر استعمال کرنے کی کوشش کرتے ہیں، تو آپ کو ایک ایرر ملے گا:

Dr. Muhammad Siddique 03006329919

```
>>> age = input("How old are you? ")
How old are you? 21
>>> age >= 18
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: '>=' not supported between instances of 'str' and '
    int'
```

جب آپ input کو عددی موازنہ کے لیے استعمال کرنے کی کوشش کرتے ہیں، تو پائتھون ایک ایرر دیتا ہے کیونکہ یہ سٹرنگ اور عدد کے درمیان موازنہ نہیں کر سکتا: جو سٹرنگ '21' age کو assign کی گئی ہے، وہ عددی value- 18 سے موازنہ نہیں کی جا سکتی۔

ہم اس مسئلے کو int () فنکشن کا استعمال کرتے ہوئے حل کر سکتے ہیں، جو input سٹرنگ کو ایک عددی value- میں تبدیل کر دیتا ہے۔ اس سے موازنہ کامیابی سے چلتا ہے:

```
>>> age = input("How old are you? ")
How old are you? 21
>>> age = int(age)
>>> age >= 18
True
```

اس مثال میں، جب ہم 21 درج کرتے ہیں، پائتھون عدد کو سٹرنگ کے طور پر interpret کرتا ہے، لیکن int() کے ذریعے وہ عددی نمائندگی میں تبدیل ہو جاتا ہے۔ اب پائتھون اس شرطی ٹیسٹ کو چلا سکتا ہے: یہ age (جواب عددی value- 21 کو ظاہر کرتا ہے) اور 18 کا موازنہ کرتا ہے تاکہ یہ چیک کرے کہ age 18 کے برابر یا اس سے زیادہ ہے۔

26.2 int() فنکشن کا استعمال ایک حقیقی پروگرام میں

آپ int() فنکشن کو ایک حقیقی پروگرام میں کیسے استعمال کرتے ہیں؟ فرض کریں کہ ایک پروگرام ہے جو یہ طے کرتا ہے کہ کیا لوگ رولر کوسٹر پر سواری کرنے کے لیے کافی لمبے ہیں:

```
rollercoaster.py
height = input("How tall are you, in inches? ")
height = int(height)
if height >= 48:
    print("\nYou're tall enough to ride!")
else:
    print("\nYou'll be able to ride when you're a little older.
    ")
```

پروگرام height کا موازنہ 48 سے کر سکتا ہے کیونکہ height = int(height) input-value کو عددی نمائندگی میں تبدیل کر دیتا ہے اس سے پہلے کہ موازنہ کیا جائے۔ اگر درج کیا گیا عدد 48 سے بڑا یا اس کے برابر ہو تو ہم User کو بتاتے ہیں کہ وہ کافی لمبا ہے:

Dr. Muhammad Siddique 03006329919

```
How tall are you, in inches? 71
You're tall enough to ride!
```

جب آپ عددی input کا استعمال حساب کتاب اور موازنہ کے لیے کرتے ہیں، تو یہ ضروری ہے کہ input کی value- کو سب سے پہلے عددی نمائندگی میں تبدیل کر لیا جائے۔

27.2 موڈیولو آپریٹر

اعدادی معلومات کے ساتھ کام کرنے کے لئے ایک مفید ٹول موڈیولو آپریٹر (%) ہے، جو ایک نمبر کو دوسرے نمبر سے تقسیم کرتا ہے اور باقی ماندہ (ری مینڈر) واپس کرتا ہے:

```
>>> 4 \% 3
1
>>> 5 \% 3
2
>>> 6 \% 3
0
>>> 7 \% 3
1
```

موڈیولو آپریٹر آپ کو یہ نہیں بتاتا کہ ایک نمبر دوسرے نمبر میں کتنی بار فٹ آتا ہے؛ یہ صرف آپ کو باقی ماندہ بتاتا ہے۔ جب ایک نمبر دوسرے نمبر سے تقسیم ہو جاتا ہے، تو باقی ماندہ 0 ہوتا ہے، لہذا موڈیولو آپریٹر ہمیشہ 0 واپس کرتا ہے۔ آپ اس حقیقت کو استعمال کر کے یہ معلوم کر سکتے ہیں کہ کوئی نمبر طاق (odd) ہے یا جفت (even):

```
even_or_odd.py
number = input("Enter a number, and I will tell you if it is
    even or odd: ")
number = int(number)
if number \% 2 == 0:
    print(f"\نمبر {number} Even")
else:
    print(f"\نمبر {number} Odd")
```

جفت نمبروں کو ہمیشہ دو سے تقسیم کیا جا سکتا ہے، اس لئے اگر کسی نمبر کا موڈیولو دو سے صفر ہو (یہاں اگر number % 2 == 0) تو نمبر جفت ہے۔ ورنہ، وہ طاق ہو گا۔

```
Enter a number, and I will tell you if it is even or odd: 42
The number 42 is even.
```

Dr. Muhammad Siddique 03006329919

While-Loop 28.2

For Loop ایک elements کے مجموعے کو لیتا ہے اور ہر آئٹم کے لئے ایک بار code کے بلاک کو انجام دیتا ہے۔ اس کے برعکس، While-Loop اس وقت تک چلتا رہتا ہے جب تک کہ ایک مخصوص شرط سچ نہ ہو۔

While-Loop کا عمل میں آنا 29.2

آپ While-Loop کا استعمال نمبروں کے سلسلے کو گننے کے لئے کر سکتے ہیں۔ مثال کے طور پر، نیچے دیا گیا While-Loop 1 سے 5 تک گنتی کرتا ہے:

```
counting.py
current_number = 1
while current_number <= 5:
    print(current_number)
    current_number += 1
```

پہلی لائن میں ہم current_number کو 1 سے شروع کرتے ہیں۔ پھر While-Loop اس بات کو یقینی بناتا ہے کہ current_number کی value 5 سے کم یا برابر ہونے تک پروگرام چلتا رہے۔ Loop کے اندر کا code current_number کی value پرنٹ کرتا ہے اور پھر اس value میں 1 کا اضافہ کرتا ہے۔ (+= آپریٹر کی شکل current_number=current_number+1 کی مختصر شکل ہے)۔

پائتھون اس Loop کو اس وقت تک دہراتا رہتا ہے جب تک کہ شرط current_number<=5 سچ ہو۔ چونکہ 1 پانچ سے کم ہے، پائتھون 1 پرنٹ کرتا ہے اور پھر 1 کا اضافہ کر کے current_number کو 2 بنا دیتا ہے۔ پھر 2 پانچ سے کم ہے، پائتھون 2 پرنٹ کرتا ہے اور 1 کا اضافہ کر کے current_number کو 3 بنا دیتا ہے، اور اسی طرح۔
جب current_number کی value- پانچ سے بڑی ہو جاتی ہے، Loop رک جاتا ہے اور پروگرام ختم ہو جاتا ہے۔

```
1
2
3
4
5
```

30.2 User کو جب چاہے پروگرام بند کرنے کی اجازت دینا

ہم پروگرام کو اس وقت تک چلنے دے سکتے ہیں جب تک User چاہے۔ ہم پروگرام کے زیادہ تر حصے کو Loop-While کے اندر رکھ سکتے ہیں۔ ہم value-quit کو ڈیفائن کریں گے اور پروگرام کو اس وقت تک چلنے دیں گے جب تک User نے value-quit نہ ڈالی ہو:

Dr. Muhammad Siddique 03006129919

```
parrot.py
prompt = "\nTell me something, and I will repeat it back to you:"
prompt += "\nEnter 'quit' to end the program. "
message = ""
while message != 'quit':
    message = input(prompt)
    print(message)
```

ہم پہلے ایک پرامپٹ ڈیفائن کرتے ہیں جو User کو ان کے دو آپشنز بتاتا ہے: پیغام داخل کرنا یا value-quit داخل کرنا (اس صورت میں۔ 'quit') پھر ہم ایک message-variable کو ڈیفائن کرتے ہیں تاکہ جو بھی User داخل کرے وہ اس میں محفوظ ہو سکے۔ ہم message کو ایک خالی سٹرنگ "" کے طور پر ڈیفائن کرتے ہیں تاکہ پائتھون کے پاس پہلا چیک کرنے کے لئے کچھ ہو۔ جب پروگرام پہلی بار چلتا ہے اور پائتھون While اسٹیٹمنٹ تک پہنچتا ہے، تو اسے message کی value- 'quit' سے موازنہ کرنا ہوتا ہے، لیکن ابھی تک User کا Input داخل نہیں ہوا ہوتا۔ اگر پائتھون کے پاس موازنہ کرنے کے لئے کچھ نہ ہو، تو یہ پروگرام کو جاری نہیں رکھ سکتا۔ اس مسئلے کو حل کرنے کے لئے ہم message کو ابتدائی value- دے دیتے ہیں۔ اگرچہ یہ صرف ایک خالی سٹرنگ ہے، لیکن یہ پائتھون کے لئے معنی خیز ہو گا اور اسے وہ موازنہ کرنے کی اجازت دے گا جو Loop-While کو کام کرنے دیتا ہے۔

اب Loop-While تب تک چلتا رہے گا جب تک message کی value- 'quit' نہ ہو۔
پہلی بار Loop کے اندر message صرف ایک خالی سٹرنگ ہے، اس لئے پائتھون Loop میں داخل ہوتا ہے۔ جب message=input(prompt) تک پہنچتا ہے، تو پائتھون پرامپٹ کو دکھاتا ہے اور User کے Input کا

انتظار کرتا ہے۔ جو بھی وہ داخل کرتا ہے، وہ message میں محفوظ ہو جاتا ہے اور پرنٹ ہو جاتا ہے۔ پھر پائتھون While اسٹیٹمنٹ میں شرط کا دوبارہ جائزہ لیتا ہے۔ جب تک User 'quit' نہیں لکھتا، پرامپٹ دوبارہ دکھایا جاتا ہے اور پائتھون مزید Input کا انتظار کرتا ہے۔ جب آخر کار User 'quit' داخل کرتا ہے، پائتھون While-Loop کو روک دیتا ہے اور پروگرام ختم ہو جاتا ہے۔

31.2 پرچم (Flag) کا استعمال

پچھلے مثال میں، ہم نے پروگرام کو ایک دی گئی شرط کے سچ ہونے تک مخصوص کام کرنے کو کہا۔ لیکن اگر پروگرام میں کئی مختلف واقعات ہیں جو پروگرام کو بند کر سکتے ہیں تو کیا ہو گا؟

مثال کے طور پر، ایک کھیل میں کئی مختلف واقعات ہیں جو کھیل کو ختم کر سکتے ہیں۔ جب کھلاڑی کے پاس جہاز ختم ہو جاتے ہیں، ان کا وقت ختم ہو جاتا ہے، یا وہ شہر جنہیں انہوں نے بچانا تھا، تب کھیل ختم ہو جاتا ہے۔ یہ کسی ایک بھی واقعہ کے ہونے پر ختم ہونا چاہیے۔ اگر کئی مختلف واقعات ہیں جو پروگرام کو بند کر سکتے ہیں، تو ان تمام شرائط کو ایک While اسٹیٹمنٹ میں آزمانا پیچیدہ ہو جاتا ہے۔

ایسی صورتحال میں، آپ ایک variable ڈیفائن کر سکتے ہیں جو پروگرام کے چلنے یا رکنے کا تعین کرے گا۔ اس variable کو پرچم (Flag) کہا جاتا ہے، جو پروگرام کے لئے ایک سگنل کی طرح کام کرتا ہے۔ ہم اپنے پروگرام کو اس طرح لکھ سکتے ہیں کہ یہ اس وقت تک چلتا رہے جب تک پرچم کی Truevalue- ہو اور رک جائے جب کسی بھی واقعے کی وجہ سے پرچم کی Falsevalue- ہو جائے۔ اس کے نتیجے میں، ہمارا While اسٹیٹمنٹ صرف ایک شرط کو چیک کرے گا: کیا پرچم کی Truevalue- ہے؟ پھر باقی پروگرام میں موجود تمام چیک (جیسے elif اسٹیٹمنٹ) کو صاف ستھرا طور پر ترتیب دیا جا سکتا ہے۔

Dr. Muhammad Siddique 03006529919

ہم parrot.py میں ایک پرچم شامل کرتے ہیں جو پروگرام کے چلنے کی حالت کو مانیٹر کرے گا:

```
prompt = "\nTell me something, and I will repeat it back to you
    :"
prompt += "\nEnter 'quit' to end the program. "
active = True
while active:
    message = input(prompt)
    if message == 'quit':
        active = False
    else:
        print(message)
```

ہم variableactive کو True سیٹ کرتے ہیں تاکہ پروگرام فعال حالت میں شروع ہو۔ اس سے While اسٹیٹمنٹ سادہ ہو جاتا ہے کیونکہ While اسٹیٹمنٹ میں کوئی موازنہ نہیں کیا جاتا؛ لاجک دوسرے حصوں میں سنبھال لی جاتی ہے۔ جب تک Truevariableactive رہتا ہے، Loop چلتا رہتا ہے۔

While-Loop کے اندر موجود if اسٹیٹمنٹ میں ہم message کی value- چیک کرتے ہیں جب User اپنا Input داخل کرتا ہے۔ اگر User 'quit' داخل کرتا ہے، تو ہم active کو False سیٹ کر دیتے ہیں اور While

Loop-رک جاتا ہے۔ اگر User کچھ اور داخل کرتا ہے تو ہم ان کے پیغام کو پرنٹ کرتے ہیں۔
یہ پروگرام پچھلے مثال کی طرح ہی کام کرتا ہے جہاں ہم نے شرط کو براہ راست While اسٹیٹمنٹ میں رکھا تھا۔ لیکن اب چونکہ ہمارے پاس پرچم موجود ہے جو یہ بتاتا ہے کہ پروگرام فعال حالت میں ہے، ہم مزید ٹیسٹ (جیسے elif اسٹیٹمنٹ) آسانی سے شامل کر سکتے ہیں تاکہ وہ واقعات جو پرچم کو False میں تبدیل کریں، کو چیک کیا جا سکے۔

32.2 break-statement

جب آپ کسی loop-while کو فوراً اس کے باقی code کو چلائے بغیر چھوڑنا چاہتے ہیں، چاہے کوئی بھی شرائطی ٹیسٹ کا نتیجہ ہو، تو آپ break اسٹیٹمنٹ کا استعمال کر سکتے ہیں۔ break اسٹیٹمنٹ آپ کے پروگرام کے بہاؤ کی سمت کو کنٹرول کرتا ہے؛ آپ اس کا استعمال کر کے یہ طے کر سکتے ہیں کہ کون سے code لائنز چلیں اور کون سی نہیں، تاکہ پروگرام صرف وہی code چلائے جو آپ چاہتے ہیں اور جس وقت آپ چاہتے ہیں۔
مثال کے طور پر، ایک پروگرام فرض کریں جو User سے پوچھتا ہے کہ وہ کون سے مقامات پر گئے ہیں۔ ہم اس پروگرام میں break کا استعمال کر کے loop-while کو روک سکتے ہیں جب User 'quit'-value درج کرے:

Dr. Muhammad Siddique 03006329919

```
cities.py
prompt = "\nPlease enter the name of a city you have visited:"
prompt += "\n(Enter 'quit' when you are finished.) "
while True:
    city = input(prompt)
    if city == 'quit':
        break
    else:
        print(f"I'd love to go to {city.-title()}!")
```

یہ loop-جو True while سے شروع ہوتا ہے، ہمیشہ کے لیے چلے گا جب تک کہ وہ break اسٹیٹمنٹ تک نہ پہنچے۔
یہ پروگرام User سے ان شہروں کے نام داخل کرنے کے لیے مسلسل پوچھتا رہے گا جن میں وہ گئے ہیں، جب تک کہ وہ 'quit' نہ درج کرے۔ جیسے ہی وہ 'quit' درج کریں گے، break اسٹیٹمنٹ چل جائے گی، جس سے loop-Python سے باہر نکل جائے گا:

```
Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) New York
I'd love to go to New York!
Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) San Francisco
I'd love to go to San Francisco!
Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) quit
```

نوٹ: آپ break اسٹیٹمنٹ کو Python کی کسی بھی loop-میں استعمال کر سکتے ہیں۔ مثال کے طور پر، آپ break کا استعمال کسی loop-for میں کر سکتے ہیں جو ایک فہرست یا ڈکشنری پر کام کر رہا ہو۔

33.2 continue-statement-میں-loop

اگر آپ loop-سے مکمل طور پر باہر نکلنا نہیں چاہتے بلکہ اس کے باقی code کو چھوڑ کر loop-کے آغاز پر واپس جانا چاہتے ہیں، تو آپ continue اسٹیٹمنٹ استعمال کر سکتے ہیں تاکہ اس کے مطابق شرائطی ٹیسٹ کے نتیجے پر loop-کے آغاز پر واپس جائیں۔

مثال کے طور پر، ایک loop-تصور کریں جو 1 سے 10 تک گنتی کرتا ہے لیکن صرف اس رینج میں فردی نمبر چھاپتا ہے:

```
counting.py
current_number = 0
while current_number < 10:
    current_number += 1
    if current_number % 2 == 0:
        continue
    print(current_number)
```

سب سے پہلے ہم current_number کو 0 پر سیٹ کرتے ہیں۔ چونکہ یہ 10 سے کم ہے، Python loop-while میں داخل ہوتا ہے۔ ایک بار جب ہم loop-میں داخل ہوتے ہیں، ہم گنتی 1 سے بڑھاتے ہیں، تو current_number 1 بن جاتا ہے۔ پھر if اسٹیٹمنٹ current_number اور 2 کے موڈیولو کو چیک کرتی ہے۔ اگر موڈیولو 0 ہوتا ہے (یعنی current_number 2 سے تقسیم ہو سکتا ہے)، تو continue اسٹیٹمنٹ Python کو باقی loop-نظر انداز کرنے کا کہتی ہے اور loop-کے آغاز پر واپس لے جاتی ہے۔ اگر current_number 2 سے تقسیم نہیں ہوتا، تو باقی loop-چلتا ہے اور Python current_number چھاپتا ہے:

Dr. Muhammad Siddique 03006329919

```
1
3
5
7
9
```

34.2 loop-infinite سے بچنا

ہر loop-while کو ختم ہونے کا ایک طریقہ ہونا چاہیے تاکہ یہ ہمیشہ کے لیے چلتا نہ رہے۔ مثال کے طور پر، یہ گنتی کرنے والا loop-1 سے 5 تک گنتی کرنی چاہیے:

```
counting.py
x = 1
while x <= 5:
    print(x)
    x += 1
```

لیکن اگر آپ x += 1 کی لائن کو بھول جائیں، تو loop-ہمیشہ کے لیے چلتا رہے گا:

```
# This loop runs forever!
x = 1
while x <= 5:
    print(x)
```

اب x کی value-1 سے شروع ہو گی لیکن کبھی تبدیل نہیں ہو گی۔ اس کے نتیجے میں، x <= 5 کا شرائطی ٹیسٹ ہمیشہ True ہو گا اور loop-while ہمیشہ چلتا رہے گا، اور ایک سلسلہ 1 پر پرنٹ ہوتا رہے گا، جیسے یہ:

```
1
1
1
1
```

ہر پروگرامر کبھی نہ کبھی ایک لامتناہی loop-while لکھتا ہے، خاص طور پر جب کسی پروگرام کی loop-س کے خارجی حالات پیچیدہ ہوں۔ اگر آپ کا پروگرام لامتناہی loop- میں پھنس جائے، تو CTRL-C دبائیں یا صرف اس ٹرمینل ونڈو کو بند کریں جس میں پروگرام کا output دکھ رہا ہو۔ لامتناہی loop-س سے بچنے کے لیے، ہر loop-while کو ٹیسٹ کریں اور اس بات کو یقینی بنائیں کہ loop- وہیں رک جائے جہاں آپ چاہتے ہیں۔ اگر آپ چاہتے ہیں کہ آپ کا پروگرام User کے داخل کردہ کسی خاص value- پر ختم ہو، تو پروگرام کو چلائیں اور وہ value- درج کریں۔ اگر پروگرام ختم نہیں ہوتا، تو اس طریقے کا بغور جائزہ لیں جس سے پروگرام اس value- کو ہینڈل کرتا ہے جو loop- کو ختم کرنا چاہیے۔ یقینی بنائیں کہ پروگرام کا کوئی حصہ کم از کم loop- کی شرط کو False بنا سکتا ہو یا اس تک break اسٹیٹمنٹ پہنچا سکتا ہو۔ نوٹ: وی ایس، code جیسے بہت سے ایڈیٹرز، output کو ایک ایمبیڈڈ ٹرمینل ونڈو میں دکھاتا ہے۔ لامتناہی loop- کو منسوخ کرنے کے لیے، اس بات کو یقینی بنائیں کہ آپ output ایریا میں کلک کریں اس سے پہلے کہ CTRL-C دبائیں۔

Dr. Muhammad Siddique 03086329919

35.2 Loop-While کے ساتھ Lists اور Dictionaries کا استعمال

اب تک ہم نے صرف ایک User کی معلومات پر کام کیا ہے۔ ہم نے User کی Input وصول key اور پھر Input یا اس کے جواب کو پرنٹ کیا۔ اگلی بار جب ہم Loop-While کے ذریعے جائیں گے، ہم ایک اور Input وصول کریں گے اور اس کا جواب دیں گے۔ لیکن جب ہمیں کئی زUser اور معلومات کو ٹریک کرنا ہو، تو ہمیں lists اور dictionaries کا استعمال Loop-sWhile کے ساتھ کرنا ہو گا۔

ایک for loop لسٹ کے ذریعے loop- کرنے کے لیے مؤثر ہے، لیکن آپ کو for loop میں لسٹ میں ترمیم نہیں کرنی چاہیے کیونکہ Python کو لسٹ میں موجود آئٹمز کو ٹریک کرنے میں مشکلات پیش آئیں گی۔ جب آپ لسٹ میں ترمیم کرنا چاہیں، تو Loop- While کا استعمال کریں۔ lists اور dictionaries کے ساتھ Loop- While کا استعمال آپ کو بہت ساری Input جمع کرنے، محفوظ کرنے اور ترتیب دینے کی اجازت دیتا ہے تاکہ بعد میں ان کا تجزیہ اور رپورٹ کی جا سکے۔

36.2 ایک لسٹ سے دوسری لسٹ میں elements منتقل کرنا

فرض کریں کہ ایک ویب سائٹ کے نئے رجسٹرڈ لیکن غیر تصدیق شدہ Users کی لسٹ ہے۔ جب ہم ان Users کی تصدیق کر لیتے ہیں تو ہم انہیں کس طرح تصدیق شدہ Users کی الگ لسٹ میں منتقل کر سکتے ہیں؟ ایک طریقہ یہ ہو سکتا ہے کہ ہم Loop-While کا استعمال کریں تاکہ جیسے ہی ہم User کی تصدیق کریں، ان کو غیر تصدیق شدہ Users کی لسٹ سے نکال کر تصدیق شدہ Users کی الگ لسٹ میں شامل کر لیں۔ code کچھ اس طرح نظر آ سکتا ہے:

```
# Start with users that need to be verified,
# and an empty list to hold confirmed users.
unconfirmed_users = ['alice', 'brian', 'candace']
confirmed_users = []

# Verify each user until there are no more unconfirmed users.
# Move each verified user into the list of confirmed users.
while unconfirmed_users:
    current_user = unconfirmed_users.pop()
    print(f"Verifying user: {current_user.-title()}")
    confirmed_users.append(current_user)

# Display all confirmed users.
print("\nThe following users have been confirmed:")
for confirmed_user in confirmed_users:
    print(confirmed_user.-title())
```

Dr. Muhammad Siddique 03006329919

ہم نے شروع میں غیر تصدیق شدہ Users کی لسٹ، (Alice، Brian اور Candace) لی اور تصدیق شدہ Users کے لیے ایک خالی لسٹ رکھی۔ loop-while اس وقت تک چلتا رہے گا جب تک unconfirmed_users کی لسٹ خالی نہیں ہو جاتی۔ اس loop کے اندر metho-dpop () غیر تصدیق شدہ User کو ایک وقت میں نکال کر current_user میں محفوظ کرتا ہے، اور انہیں تصدیق شدہ User کی لسٹ میں شامل کر دیتا ہے۔ جیسے ہی غیر تصدیق شدہ User کی لسٹ ختم ہوتی ہے، تصدیق شدہ User کی لسٹ کو پرنٹ کیا جاتا ہے۔

37.2 ایک لسٹ سے مخصوص values کو ہٹانا

چند دن پہلے ہم نے method-remove () کا استعمال کیا تھا تاکہ کسی مخصوص value- کو لسٹ سے نکال سکیں۔ یہ method اس وقت کام کرتا ہے جب جو value- ہم ہٹانا چاہتے ہیں وہ صرف ایک بار لسٹ میں موجود ہو۔ لیکن اگر آپ کو کسی value- کی تمام مثالیں لسٹ سے نکالنی ہوں تو کیا ہو گا؟

فرض کریں کہ آپ کے پاس پالتو جانوروں کی ایک لسٹ ہے جس میں value-'cat' کئی بار آئی ہے۔ اس value- کو تمام مثالوں سے ہٹانے کے لیے آپ Loop- While کا استعمال کر سکتے ہیں جب تک کہ 'cat' لسٹ میں نہ ہو، جیسے کہ یہاں دکھایا گیا ہے:

```
pets = ['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat'
    ]
```

```
print(pets)
while 'cat' in pets:
    pets.remove('cat')
print(pets)
```

ہم نے شروع میں 'cat'-value کی کئی مثالوں کے ساتھ ایک لسٹ رکھی۔ لسٹ پرنٹ کرنے کے بعد Python Loop-While میں داخل ہو گا کیونکہ وہ 'cat'-value کو کم از کم ایک بار لسٹ میں پائے گا۔ جب وہ loop میں جائے گا، وہ 'cat' کی پہلی مثال کو ہٹا دے گا اور پھر دوبارہ loop میں واپس جائے گا جب تک کہ لسٹ سے تمام 'cat' values نہیں ہٹ جاتیں۔

38.2 InputUser کے ساتھ ڈکشنری بھرنا

آپ ہر بار Loop- While کے ذریعے جتنی چاہیں Input لے سکتے ہیں۔ آیئے ایک پولنگ پروگرام بناتے ہیں جس میں ہر بار Loop- While کے ذریعے ہم User کا نام اور جواب وصول کرتے ہیں۔ ہم جو ڈیٹا جمع کرتے ہیں اسے ایک ڈکشنری میں محفوظ کریں گے کیونکہ ہم ہر جواب کو ایک خاص User سے جوڑنا چاہتے ہیں۔

```
responses = {}

# Polling is active
polling_active = True
while polling_active:
    # Ask for the person's name and response.
    name = input("\nWhat is your name? ")
    response = input("Which mountain would you like to climb someday? ")

    # Store the response in the dictionary.
    responses[name] = response

    # Find out if anyone else wants to respond to the poll.
    repeat = input("Would you like to let another person respond? (yes/ no) ")
    if repeat == 'no':
        polling_active = False

# Polling is complete. Display the results.
print("\n--- Poll Results ---")
for name, response in responses.items():
    print(f"{name} would like to climb {response}.")
```

Dr. Muhammad Siddique 03006329919

پروگرام سب سے پہلے ایک خالی ڈکشنری (responses) کی declare کرتا ہے اور پولنگ کے عمل کے فعال ہونے کا اشارہ دینے کے لیے ایک فلیگ (polling_active) سیٹ کرتا ہے۔ جب تک polling_active True ہے، Python Loop- While میں موجود code کو چلائے گا۔

اس loop-کے اندر، User سے ان کا نام اور جس پہاڑ کو وہ چڑھنا چاہتے ہیں پوچھا جائے گا۔ وہ معلومات responses ڈکشنری میں محفوظ ہو گی، اور پھر User سے پوچھا جائے گا کہ کیا وہ کسی اور کو پول میں جواب دینے کی اجازت دینا چاہتے ہیں۔

اگر وہ 'yes' جواب دیتے ہیں تو پروگرام دوبارہ While -Loop میں داخل ہوگا۔ اگر وہ 'no' جواب دیتے ہیں، تو پولنگ کا عمل ختم ہو جائے گا اور نتائج پرنٹ کیے جائیں گے۔

خلاصہ

اس باب میں آپ نے سیکھا کہ کس طرح input() کا استعمال کر کے آپ اپنے پروگرامز میں Users کو اپنی معلومات فراہم کرنے کی اجازت دے سکتے ہیں۔ آپ نے متنی اور عددی Input دونوں کے ساتھ کام کرنا سیکھا اور یہ بھی سیکھا کہ While Loop–س کا استعمال کس طرح کیا جائے تاکہ آپ کا پروگرام اس وقت تک چلتا رہے جب تک آپ کے زUser چاہیں۔ آپ نے While–Loop کے بہاؤ کو کنٹرول کرنے کے مختلف طریقے دیکھے، جیسے کہ ایک فعال پرچم سیٹ کرنا، break سٹیٹمنٹ کا استعمال، اور continue سٹیٹمنٹ کا استعمال۔ آپ نے سیکھا کہ کس طرح While–Loop کا استعمال کرتے ہوئے ایک لسٹ سے دوسری لسٹ میں elements کو منتقل کیا جا سکتا ہے اور کس طرح ایک value- کی تمام مثالیں لسٹ سے ہٹائی جا سکتی ہیں۔ آپ نے یہ بھی سیکھا کہ کس طرح While Loop–س کو ڈکشنریز کے ساتھ استعمال کیا جا سکتا ہے۔ باب 8 میں آپ فنکشنز کے بارے میں سیکھیں گے۔ فنکشنز آپ کو اپنے پروگرامز کو چھوٹے حصوں میں تقسیم کرنے کی اجازت دیتے ہیں، جن میں سے ہر ایک مخصوص کام کرتا ہے۔ آپ ایک فنکشن کو جتنی بار چاہیں کال کر سکتے ہیں، اور آپ اپنے فنکشنز کو الگ فائلوں میں اسٹور کر سکتے ہیں۔ فنکشنز کا استعمال کر کے آپ زیادہ موثر code لکھ سکیں گے جو حل کرنے اور دیکھ بھال کرنے میں آسان ہوگا اور مختلف پروگرامز میں دوبارہ استعمال کیا جا سکے گا۔

Dr. Muhammad Siddique 03006229919