PySpark

Python for Spark

Agenda

01 Introduction to PySpark

02 Why Python for Spark?

03 Introduction to Python

04 Values, Types, and Variables

05 Operands and Expressions

06 Conditional Statements

07 Looping Constructs

08 Python Files I/O Functions

09 Strings and Operations

10 Tuples and Operations

Introduction to **PySpark**

What is PySpark?

Definition PySpark is an **API** written in Python to support **Apache Spark**

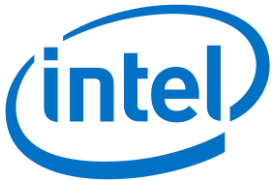
- Apache Spark is a **distributed framework** that can handle Big Data Analysis

Prerequisites for Learning PySpark

- Programming knowledge using **Python**
- Big Data knowledge and basic understanding of frameworks such as **Spark**
- One who wants to work with **Big Data** is the most suitable candidate for **PySpark**

Who uses PySpark?

Almost every firm that has access to a **huge** repository had their fair share of **problems** with **data**!



Why Python for Spark?

Easy to Learn

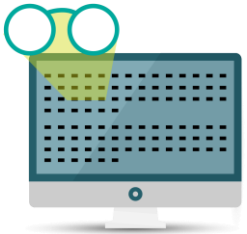
- For programmers, **Python** is comparatively **easier** to learn because of its syntax and standard libraries
- It's a **dynamically typed language** that makes it easier to hold objects of different types

Why Python for Spark?

Lots of Libraries

- Scala does **not** have enough Data Science tools and libraries like **Python** for Machine Learning and Natural Language Processing
- Scala **lacks** good visualization and local data transformations

Why Python for Spark?



Good Readability

- Easier to learn because the code is **less verbose** and **more reliable** than Scala
- With Python, **type checking** happens at runtime. This ensures that you work **quicker** without having to specify types every time

Why Python for Spark?

Large Community

- Python has one of the **biggest communities** for a programming language
- It has a **global** community comprising **millions** of learners and developers who interact online

Introduction to **Python**

Introduction to Python

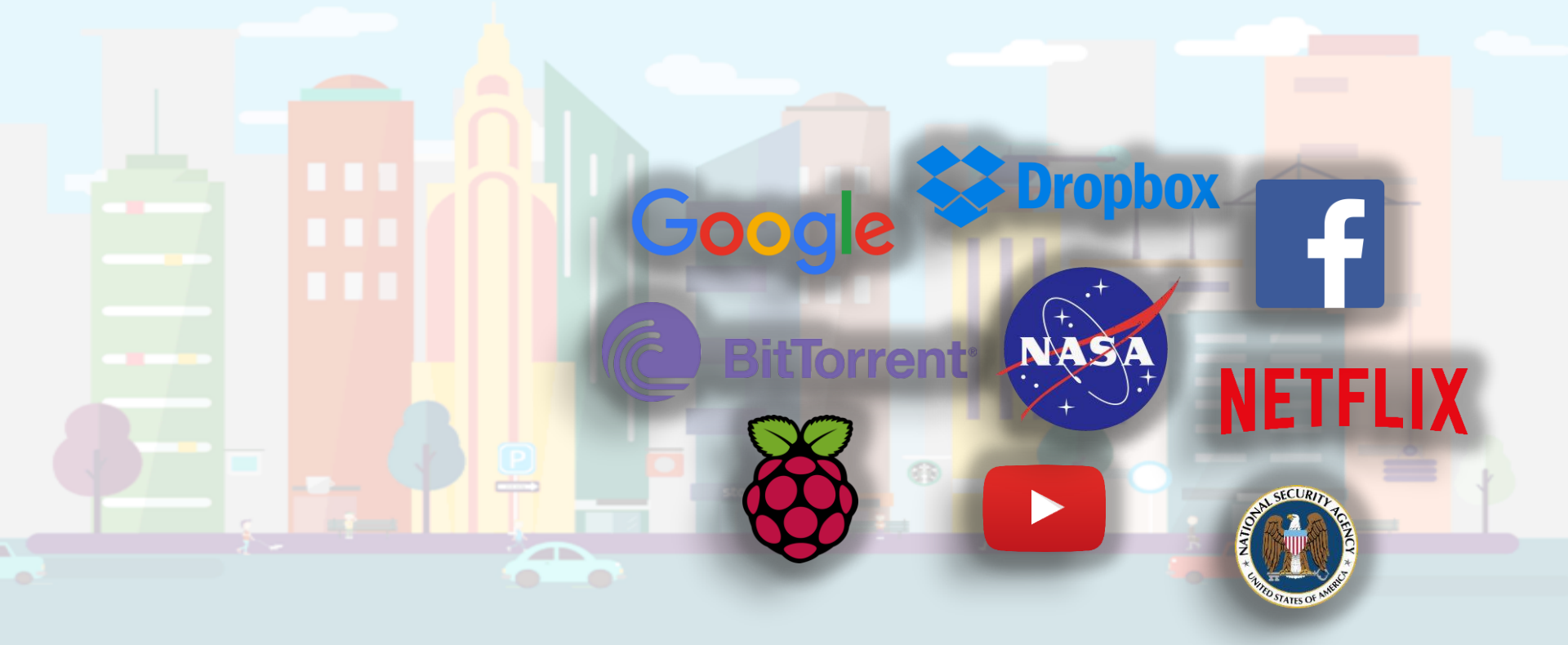
Ever wondered why it is called Python?

Named after **Monty's Python Flying Circus**

Created in 1991 by **Guido van Rossum**

Why should you learn Python?

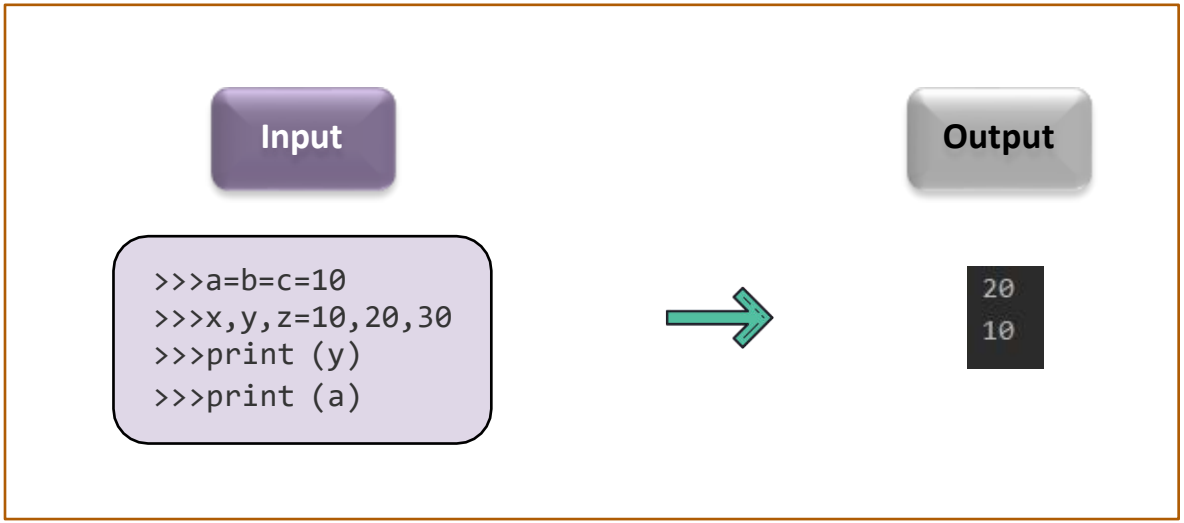
Popularity of Python



Values, Types, and Variables

Python Variables

Assigning multiple values to a variable:



Python Keywords

- **Special** reserved words
- Convey a special meaning to the **compiler/interpreter**
- Each keyword with a special meaning and a **specific** operation
- **NEVER** used as a variable!

True	False	None	and	as
asset	def	class	continue	break
else	finally	elif	del	except
global	for	if	from	import
raise	try	or	return	pass
nonlocal	in	not	is	lambda

Python Literals

Literals are the constants used in Python!

Different types of literals in Python

- String Literals
- Numeric Literals
- Boolean Literals
- Special Literals

Python Literals

- Formed by enclosing a text in **quotes**
- Both **single** and **double** quotes can be used

- **String Literals**
- Numeric Literals
- Boolean Literals
- Special Literals

Input

```
name1='John'
name2='James'
print(name1)
print(name2)
text1='hello\World'
multiline='''st1
...st2
...st3'''
print(multiline)
```

Output

```
John
James
st1
...st2
...st3
```

Python Literals

- String Literals
- **Numeric Literals**
- Boolean Literals
- Special Literals

Int	Long	Float	Complex
+ve and –ve numbers (integers) with no fractional part E.g.: 100, –234	Unlimited integer size followed by upper or lowercase L E.g.: 233424243L	Real numbers with both integer and fractional part E.g.: –213.3	In the form of **a+bj**; '**a**' forms the real part and '**b**' forms the imaginary part E.g.: 3.14j

- In Python, the value of an integer is **not restricted** by the number of bits and can expand to the **limit** of the available **memory**
- **No special arrangement** is required for storing large numbers

Python Literals

Numbers:

- **Complex numbers** are written in the form, **a + bj**, where **a** is the real part and **b** is the imaginary part

- We can use the **type() function** to know which class a variable or a value belongs to and the **isinstance() function** to check if it belongs to the particular class

Input

```
a = 5
# Output: <class 'int'>
print(type(a))
# Output: <class 'float'>
print(type(5.0))
# Output: (8+3j)
c = 5 + 3j
print(c + 3)
# Output: True
print(isinstance(c, complex))
```

Output

```
<class 'int'>
<class 'float'>
(8+3j)
True
```

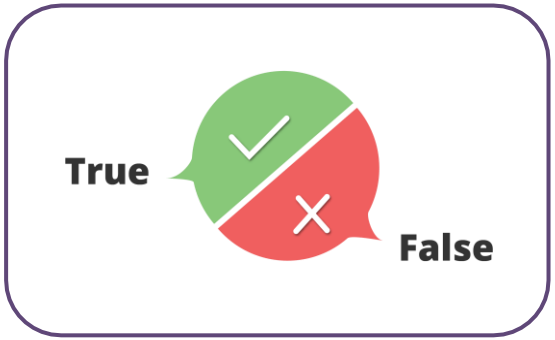
Python Literals

String Literals

Numeric Literals

Boolean Literals

Special Literals



- The bool() method in general takes only **one** parameter, on which the **standard truth testing** procedure can be applied
- If **no** parameter is passed, then by **default** it returns **False**

Python Literals

- Python has one special literal: **None**
- Used to specify the **field** that is **not** created

String Literals

Numeric Literals

Boolean Literals

Special Literals

Example

```
val1=10
val2=None
print(val1)
print(val2)
```

Output

```
10
None
```

Operators in **Python**

Types of Operators

- Arithmetic Operator
- Assignment Operator
- Comparison Operator
- Logical Operator
- Bitwise Operator
- Identity Operator
- Membership Operator

Types of Operators

- **Arithmetic Operator**
- Assignment Operator
- Comparison Operator
- Logical Operator
- Bitwise Operator
- Identity Operator
- Membership Operator

Operators

`+, -, *, /, %`

Example

```
>>> 1+2
3
>>> 1-2
-1
>>> 2%1
0
```

Types of Operators

- Arithmetic Operator
- **Assignment Operator**
- Comparison Operator
- Logical Operator
- Bitwise Operator
- Identity Operator
- Membership Operator

Operators

`=, +=, -=, *=`

Example

```
a=10
a*=10
b=a*5
print(a)
print(b)
```

Output

```
100
500
```

Types of Operators

- Arithmetic Operator
- Assignment Operator
- **Comparison Operator**
- Logical Operator
- Bitwise Operator
- Identity Operator
- Membership Operator

Operators

`>, <, >=, <=`

Example

```
a=10
b=11
if (a<b)
    print('a is lesser')
```

Output

```
a is lesser
```

Types of Operators

- Arithmetic Operator
- Assignment Operator
- Comparison Operator
- **Logical Operator**
- Bitwise Operator
- Identity Operator
- Membership Operator

Operators

`and, or , not`

Example

```
isSunday = True
isHoliday = False

if isHoliday or isSunday:
    print('Sunday is a fun day!!')
else:
    print('Not a holiday!')
```

Output

```
Sunday is a fun day!!
```

Types of Operators

- Arithmetic Operator
- Assignment Operator
- Comparison Operator
- Logical Operator
- **Bitwise Operator**
- Identity Operator
- Membership Operator

Operators

```
&, |, >>, <<, ~
```

How is it calculated?

```
111          7
101          5
111          7
```

Example

```
>>>7 | 5
7
>>>7&5
5
```

Types of Operators

- Arithmetic Operator
- Assignment Operator
- Comparison Operator
- Logical Operator
- Bitwise Operator
- **Identity Operator**
- Membership Operator

Operators

`is, is not`

Example

```
>>>7 | 5
7>>>x = 10
>>>x is 10
True

>>>x = 10
>>>x is not 10
False

>>>7&5
5
```

Types of Operators

- Arithmetic Operator
- Assignment Operator
- Comparison Operator
- Logical Operator
- Bitwise Operator
- Identity Operator
- **Membership Operator**

Operators

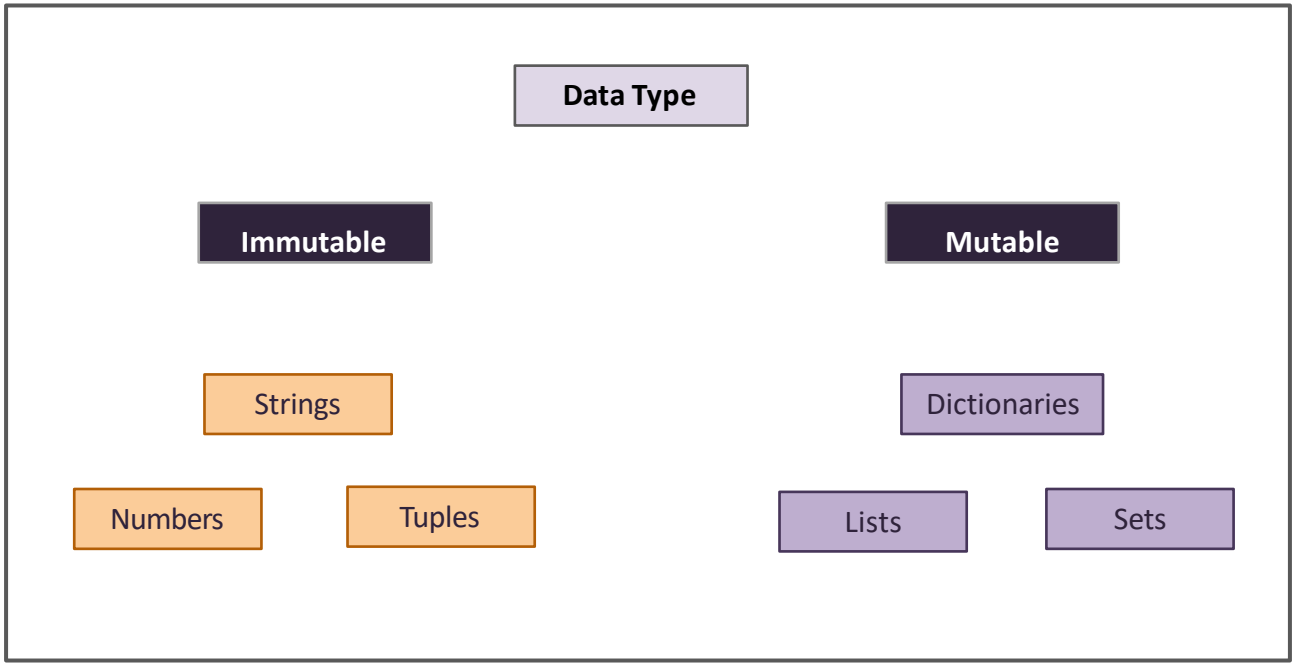
`in, not in`

Example

```
>>>pets=['dog','cat','wolf']
>>>'lion' in pets
False
>>>'wolf' in pets
True
>>>'me' in 'appointment'
True
```

Data Types in **Python**

Different Data Types in Python



Different Data Types in Python

- **Numbers**
- Strings
- Tuples
- Lists
- Dictionaries
- Sets

Example

```
message = 'Hello World'
num = 1234
rate = 13.6
print(type(message)) #return a string
print(type(num)) #return an integer
print(type(rate)) #return a float
```

Output

```
<class 'str'>
<class 'int'>
<class 'float'>
```

Different Data Types in Python

- Numbers
- **Strings**
- Tuples
- Lists
- Dictionaries
- Sets

Example

```
var1 = 'Hello-World!'
var2 = 'PythonTutorial’

print(var1[0]) # prints the first character in the string `H`
print(var2[1:5]) # prints the substring ‘ytho`
```

Output

```
H
ytho
```

String Methods

- find()
- replace()
- split()
- count()

Different Data Types in Python

- Numbers
- Strings
- **Tuples**
- Lists
- Dictionaries
- Sets

Example

```
>>>myGroup = (‘a’, ‘b', ‘c’, ‘d’)
```

Tuple Operations

- Slicing
- Indexing
- Repetition

Different Data Types in Python

- Numbers
- Strings
- Tuples
- **Lists**
- Dictionaries
- Sets

Example

```
>>>myGroup = (‘a’, 10, 7.12, ‘data’)
```

List Operations

- Concatenation
- Slicing
- Repetition
- Insert

Different Data Types in Python

- Numbers
- Strings
- Tuples
- Lists
- **Dictionaries**
- Sets

Example

```
myDict = { 1: 'John' , 2: 'Bob', 3: 'Alice' }
```

Key

Value

Dictionary Concepts

- Empty Dictionary
- Integer Keys
- Mixed Keys
- Accessing Dictionary

Different Data Types in Python

- Numbers
- Strings
- Tuples
- Lists
- Dictionaries
- **Sets**

Example

```
mySet = {1, 2, 3}
```

Set Operations

- Creating Sets
- Union
- Intersection
- Difference

Flow Control in **Python**

Flow Control Statements

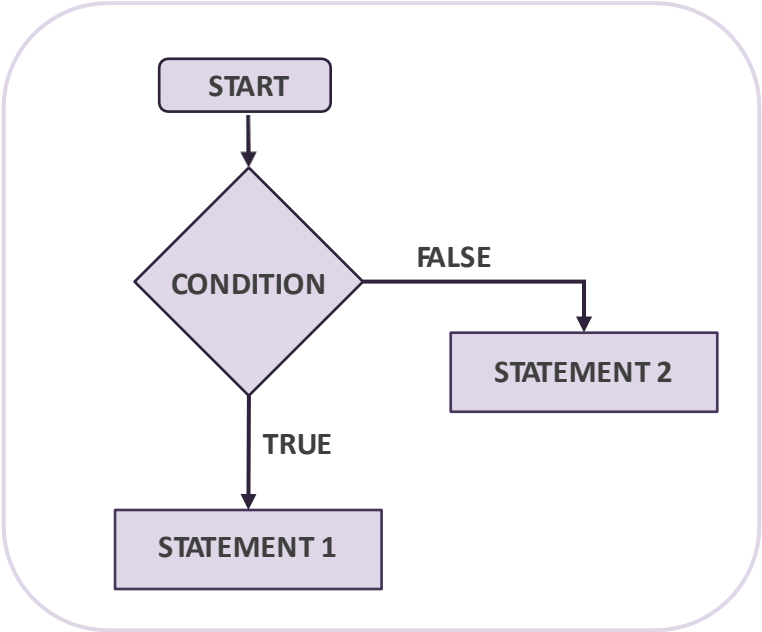
- if-else
- nested if-else
- for
- while
- break
- continue

Flow Control Statements

- **if-else**
- nested if-else
- for
- while
- break
- continue

Syntax

```
if (condition):
        statements 1 …
    else:
        statements 2 …
```

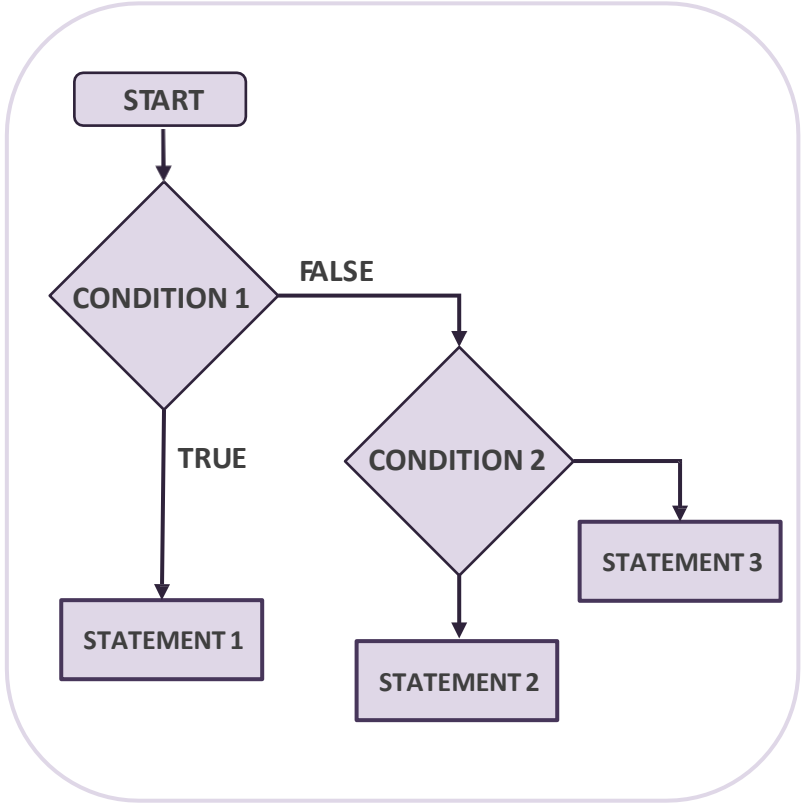


Flow Control Statements

- if-else
- **nested if-else**
- for
- while
- break
- continue

Syntax

```
if (condition 1):
        statements 1 …
    elif (condition 2):
        statements 2 …
    else
         statements 3 …
```

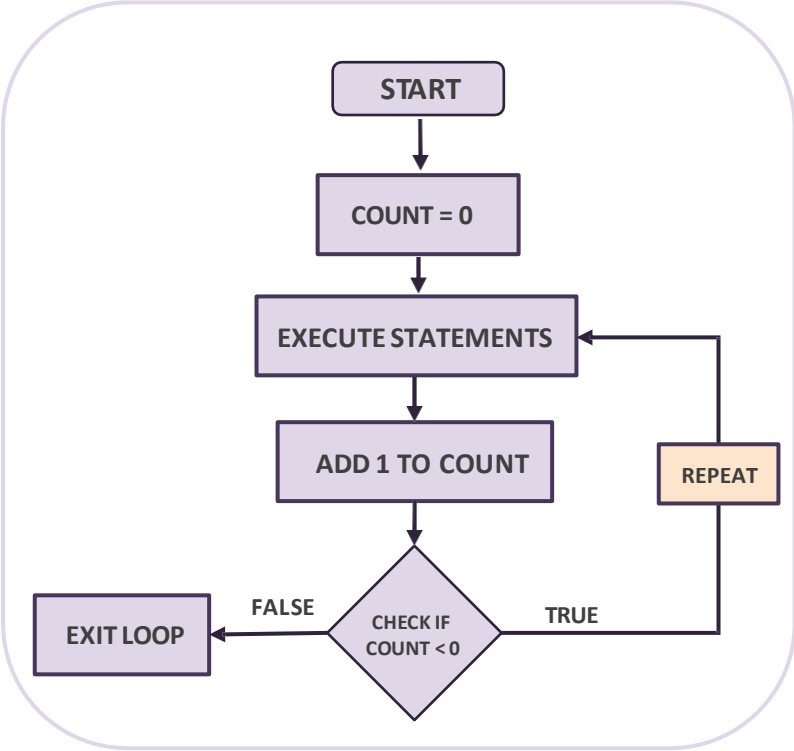


Flow Control Statements

- if-else
- nested if-else
- **for**
- while
- break
- continue

Syntax

```
for iterating_var in sequence:
execute statements
```



Flow Control Statements

- if-else
- nested if-else
- **for**
- while
- break
- continue

Code

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

Variables

```
x in fruit[0] = apple
x in fruit[1] = banana
x in fruit[2] = cherry
```

Output

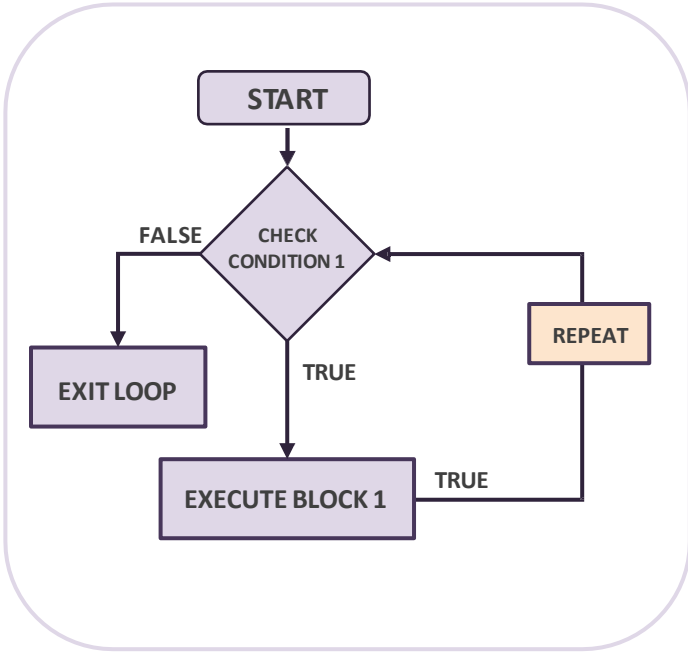
```
apple
banana
cherry
```

Flow Control Statements

- if-else
- nested if-else
- for
- **while**
- break
- continue

Syntax

```
while (condition is True):
    statements1…
```



Flow Control Statements

- if-else
- nested if-else
- for
- **while**
- break
- continue

Code

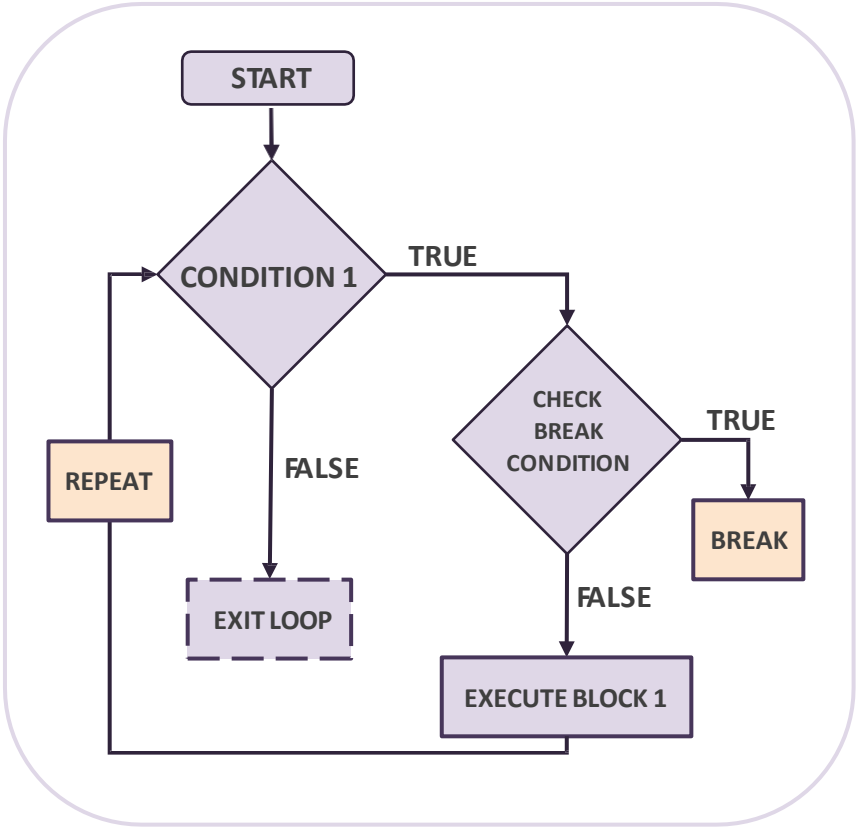
```
i = 1
while i < 6:
  print(i)
  i += 1
```

Output

```
1
2
3
4
5
```

Flow Control Statements

- if-else
- nested if-else
- for
- while
- **break**
- continue



Flow Control Statements

- if-else
- nested if-else
- for
- while
- **break**
- continue

Code

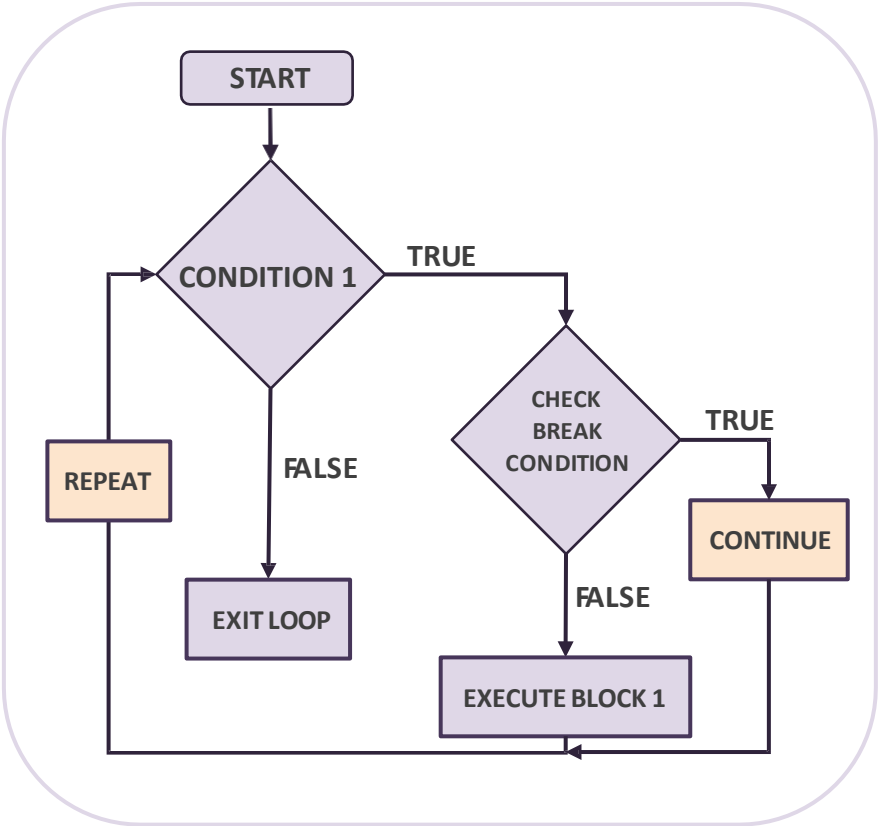
```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

Output

```
1
2
3
```

Flow Control Statements

- if-else
- nested if-else
- for
- while
- break
- **continue**



Flow Control Statements

- if-else
- nested if-else
- for
- while
- break
- **continue**

Code

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

Output

```
1
2
4
5
6
```

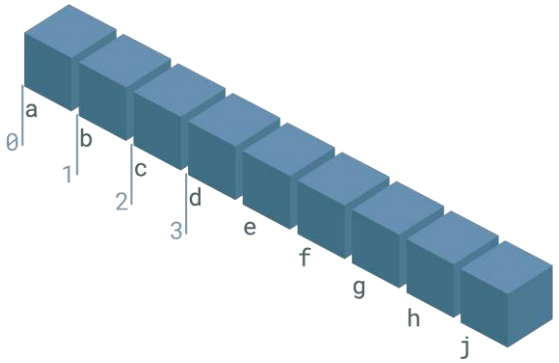
Arrays in **Python**

What is an Array?

Used to store multiple values in one single variable

Storing multiple variables

```
car1 = "Ford";
car2 = "Volvo";
car3 = "BMW";
```



Python does not have built-in support for arrays, but Python lists can be used instead

Important Array Concepts

Accessing Elements

```
>>>x = cars[0]
Ford
```

Length of the Array

```
>>>x = len(cars)
3
```

Adding the Elements

```
>>>cars.append(“Opel")]
>>>print(len(cars))
4
```

Modifying the Elements

```
>>>cars[0] = “Honda”
>>>print(cars[0])
Honda
```

Looping the Array

```
for x in cars:
  print(x)
```

Removing One Element

```
>>>cars.pop(1)
```

Removing a Specific Element

```
cars.remove("Volvo")
```

File Handling in **Python**

File Handling: Open()

The **open()** function takes two parameters: **filename** and **mode**

- **Open**
- Read
- Write/Create
- Delete

- **"r" - Read**: The default value. Opens a file for reading; returns an error if the file does not exist
- **"a" - Append**: Opens a file for appending; creates the file if it does not exist
- **"w" - Write**: Opens a file for writing; creates the file if it does not exist
- **"x" - Create**: Creates the specified file; returns an error if a file with the same name exists

Syntax

```
f = open("path of file")
```

Mode Example

```
f = open("path of file","rt")
```

File Handling: Read()

- Open
- **Read**
- Write/Create
- Delete

Example

```
f = open("demofile.txt", "r")
print(f.read())
```

Reading Parts of the File

```
f = open("demofile.txt", "r")
print(f.read(5)) #Return the 5 first
characters of the file
```

Reading the First Line

```
f = open("demofile.txt", "r")
print(f.readline()) #readline() is used
to return one line
```

File Handling: Read()

- Open
- **Read**
- Write/Create
- Delete

Reading the First Line

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline()) # using readline()
twice prints first two line
```

Loop Through the File

```
#Read the file line by line
f = open("demofile.txt", "r")
for x in f:
  print(x)
```

File Handling: Write()

To write into an existing file, you must add a parameter to the open() function

- **"a" - Append**: Appends to the end of the file
- **"w" - Write**: Overwrites any existing content

Open

Read

Write/Create

Delete

Example for Append

```
f = open("demofile.txt",  "a")
f.write("Now the file has one more line!")
```

Example for Overwrite

```
f = open("demofile.txt",  "w")
f.write("Woops! I have deleted the content!")
```

File Handling: Write()

Open

Read

Write/Create

Delete

Creating a new file:

- **"x" - Create**: Creates a file; returns an error if the file exist
- **"a" - Append**: Appends data to the end of an existing file
- **"w" - Write**: Creates a file if the specified file does not exist

Creating a File

```
f = open("myfile.txt", "x")
```

File Handling: Delete()

Importing OS module to delete a file:

- Open
- Read
- Write/Create
- **Delete**

Deleting a File

```
import os
os.remove("demofile.txt")
```

Strings in **Python**

Python Strings

String Literals

They are surrounded by either single quotation marks or double quotation marks

Assigning String to a Variable

Example

```
a = "Hello"
print(a)
```

Multiline Strings

Example

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

Python Strings

Slicing

- You can return a range of characters by using the slice syntax
- You have to specify the start index and the end index, separated by a colon, to return a part of the string

Example

```
b = "Hello, World!"
print(b[2:5])
```

Output

```
llo
```

Negative Indexing

Example

```
b = "Hello, World!"
print(b[-5:-2])
```

Output

```
orl
```

String Methods

upper()

```
a = "Hello Intellipaat!"
print(a.upper())
```

```
HELLO INTELLIPAAT!
```

lower()

```
a = "Hello Intellipaat!"
print(a.lower())
```

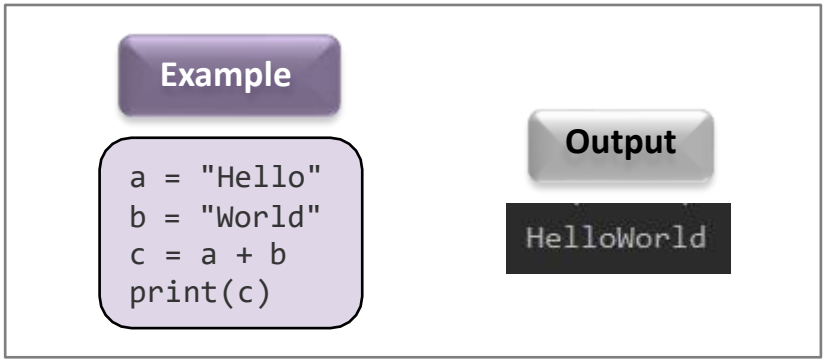
```
hello intellipaat!
```

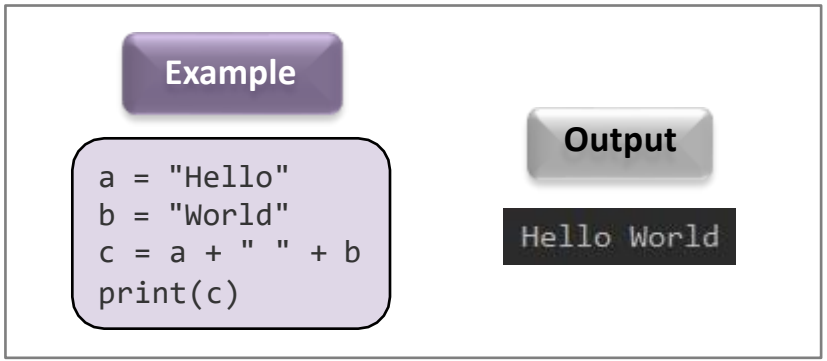
replace()

```
a = “Jello Intellipaat!"
print(a.replace(“J", “H"))
```

```
Hello Intellipaat!!!
```

String Concatenation





Collections in **Python**

Collections

There are **four** collection data types in Python:

Tuple is a collection which is ordered and unchangeable. It allows duplicate members

List is a collection which is ordered and changeable. It allows duplicate members

Dictionary is a collection which is unordered, changeable, and indexed but does not allow duplicate members

Set is a collection which is unordered and unindexed. It does not allow duplicate members

Tuples in **Python**

Tuples

- A tuple is a collection which is **ordered** and **unchangeable**
- In Python, tuples are written with **round** brackets

Example

```
thistuple = ("Python", "Java", "R")
print(thistuple)
```

Output

```
('Python', 'Java', 'R')
```

Access Tuple Items

```
thistuple = ("Python", "Java", "R")
print(thistuple[1])
```

Output

```
Java
```

Tuples

Once a tuple is created in Python, the value **cannot** be changed!

Check Item

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

Output

```
Yes, 'R' is in the fruits tuple
```

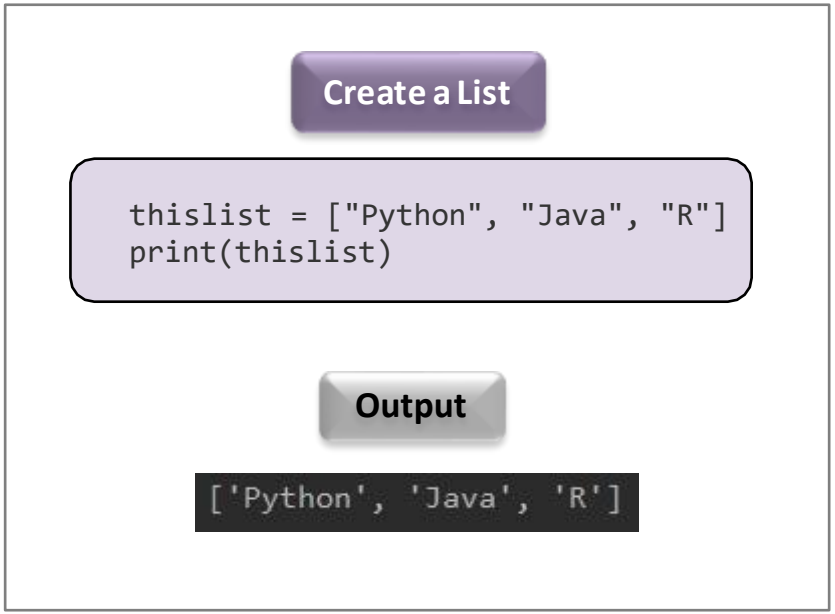
Tuple Methods

- **count()**: Returns the number of times a specified value occurs in a tuple
- **index():** Searches the tuple for a particular value and returns its respective position

Lists in **Python**

Lists

- A **list** is a collection which is ordered and changeable
- It allows duplicate members



Lists

Access Items

```
thislist = ["Python", "Java", "R"]
print(thislist[2])
```

Output

```
R
```

Change the Item Value

```
thislist = ["Python", "Java", "R"]
thislist[1] = "Scala"
print(thislist)
```

Output

```
['Python', 'Scala', 'R']
```

Lists

Loop Through a List

```
thislist = ["Python", "Java", "R"]
for x in thislist:
  print(x)
```

Output

```
Python
Java
R
```

Check if the Item Exists

```
thislist = ["Python", "Java", "R"]
if "Python" in thislist:
  print("Yes, 'Python' is in the list")
```

Output

```
Yes, 'Python' is in the list
```

Lists

List Length

```
thislist = ["Python", "Java", "R"]
print(len(thislist))
```

Output

```
3
```

Add Items

```
thislist = ["Python", "Java", "R"]
thislist.append("ReactJS")
print(thislist)
```

Output

```
['Python', 'Java', 'R', 'ReactJS']
```

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

Output

```
['Python', 'Scala', 'Java', 'R']
```

Lists

Remove Item

```
thislist = ["Python", "Java", "R"]
thislist.remove("Java")
print(thislist)
```

Output

```
['Python', 'R']
```

Remove Using pop()

```
thislist = ["Python", "Java", "R"]
thislist.pop()
print(thislist)
```

Output

```
['Python', 'Java']
```

Lists

Copy List

```
thislist = ["Python", "Java", "R"]
mylist = thislist.copy()
print(mylist)
```

Output

```
['Python', 'Java', 'R']
```

Copy Using list()

```
thislist = ["Python", "Java", "R"]
mylist = list(thislist)
print(mylist)
```

Output

```
['Python', 'Java', 'R']
```

Lists

Method	Description
append()	Adds an element at the end of a list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Adds the elements of a list to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element from the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Dictionaries in **Python**

Dictionary

- A dictionary is a collection that is unordered, changeable, and indexed
- In Python, dictionaries are written with curly brackets, and they have keys and values

Create a Dictionary

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Output

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Dictionary

Change Values

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
print(thisdict)
```

Output

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
```

Print Values

```
for x in
thisdict.values():
  print(x)
```

Output

```
Ford
Mustang
2018
```

Dictionary

Check the Key Presence

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys
in the thisdict dictionary")
```

Output

```
Yes, 'model' is one of the keys in the thisdict dictionary
```

Print the Length

```
print(len(thisdict))
```

Output

```
3
```

Dictionary

Adding Items

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

Output

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

Dictionary

Removing Items

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

Output

```
{'brand': 'Ford', 'year': 1964}
```

Dictionary

Nested Dictionaries

```
myparent = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
print(myparent)
```

Output

```
{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year': 2007}, 'child3': {'name': 'Linus', 'year': 2011}}
```

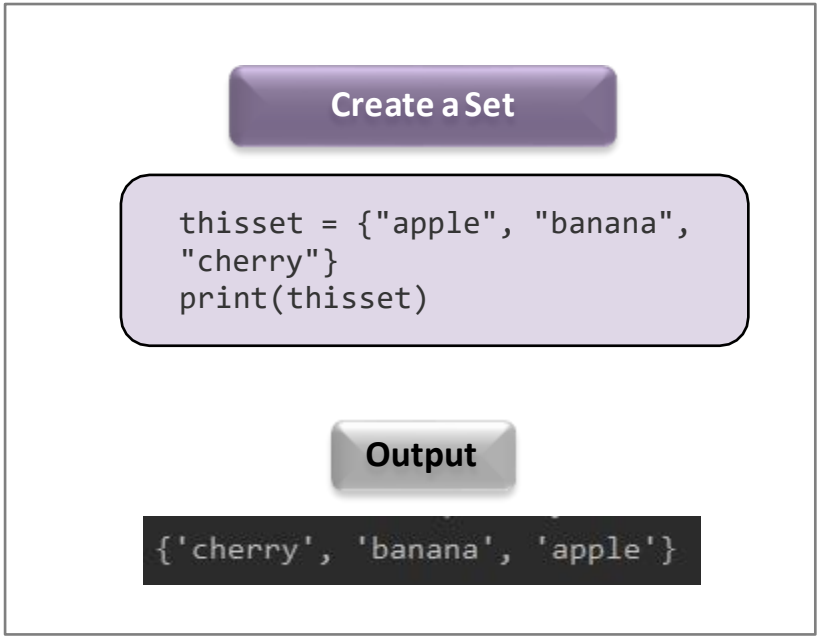
Dictionary

Method	Description
fromkeys()	Returns a dictionary with the specified keys and values
clear()	Removes all elements from the dictionary
copy()	Returns a copy of the dictionary
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key–value pair
keys()	Returns a list containing the dictionary's keys
popitem()	Removes the last inserted key–value pair
pop()	Removes the element with the specified key
setdefault()	Returns the value of the specified key. If the key does not exist, inserts the key with the specified value
update()	Updates the dictionary with the specified key–value pairs
values()	Returns a list of all values in the dictionary

Sets in **Python**

Sets

- A **set** is a collection which is unordered and unindexed
- In Python, sets are written with curly brackets



Sets

Access Items

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

Output

```
apple
cherry
banana
```

Check the Item Presence

```
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

Output

```
True
```

Sets

Add Items

```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

Output

```
{'banana', 'apple', 'cherry', 'orange'}
```

Sets

Length of a Set

```
thisset = {"apple", "banana", "cherry"}

print(len(thisset))
```

Output

```
3
```

Remove an Item

```
thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

print(thisset)
```

Output

```
{'cherry', 'apple'}
```

Sets

Method	Description
Add()	Adds an element to the set
Clear()	Removes all the elements from the set
Copy()	Returns a copy of the set
Difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes items, already included in another specified set, from the present set
discard()	Removes the specified item
intersection()	Returns a set, i.e., the intersection of two other sets
intersection_update()	Removes items, not present in other specified set(s), from the present set
isdisjoint()	Returns whether two sets have an intersection or not
issubset()	Returns whether another set contains the present set or not
issuperset()	Returns whether the present set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets

Quiz

Question 1

Where did the name ‘Python’ come from?

A	Python Snake
B	A Pet Toy
C	Monty Python’s Flying Circus
D	None of the above

Answer 1

Where did the name ‘Python’ come from?

A	Python Snake
B	A Pet Toy
C	**Monty Python’s Flying Circus**
D	None of the above

Question 2

Which among the following can be an identifier in Python?

A	@HEllo
B	1hello
C	$Hello1
D	_Hello1

Answer 2

Which among the following can be an identifier in Python?

A	@HEllo
B	1hello
C	$Hello1
D	**_Hello1**

Question 3

What is the output of the following?: 4 | 6

A	4
B	6
C	10
D	1

Answer 3

What is the output of the following?: 4 | 6

A	4
B	**6**
C	10
D	1

Question 4

What is the output of the following?: 10<<1

A	10
B	1
C	20
D	None of the above

Answer 4

What is the output of the following?: 10<<1

A	10
B	1
C	**20** ✓
D	None of the above

+91-9030485102

rganesh0203@gmail.com

https://topmate.io/rganesh_0203/