Python-Basics

Presented by:
- Ashwani Rathee
- Raag Bhutani

Python Introduction

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

1. Python Install

Many PCs and Macs will have python already installed.

To check if you have python installed on a Windows PC, search in the start bar for Python or run the following on the Command Line (cmd.exe):

- **python --version**

2. Python Syntax

Create a python file on the server, using the .py file extension, and run it in the Command Line:

- **python myfile.py**

Python Indentation

- Indentation refers to the spaces at the beginning of a code line.
- Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.
- Python uses indentation to indicate a block of code.

```python
#Correct Indendation
if 7 > 2:
  print("Seven is greater than two!")
```

Seven is greater than two!

```python
#Incorrect Indendation
if 7 > 2:
print("Seven is greater than two!")
```

```
  File "/tmp/ipykernel_33/3701325427.py", line 7
    print("Seven is greater than two!")
        ^
IndentationError: expected an indented block
```

3. Python Comments

- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.

print(“Welcome to the world of Python Programming") **#This is a comment**

4. Variables

Variables are containers for storing data values.

Creating Variables

- Python has no command for declaring a variable.
- A variable is created the moment you first assign a value to it.

Casting

- If you want to specify the data type of a variable, this can be done with casting.

Get the Type

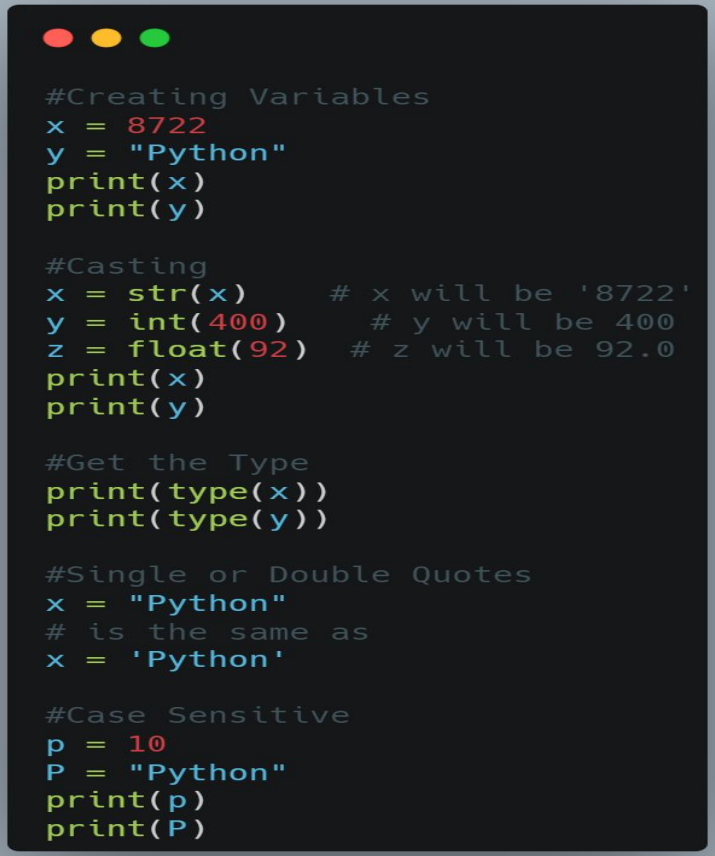
- You can get the data type of a variable with the type() function.

Single or Double Quotes

- String variables can be declared either by using single or double quotes

Case-Sensitive

- Variable names are case-sensitive.



```python
#Creating Variables
x = 8722
y = "Python"
print(x)
print(y)

#Casting
x = str(x)    # x will be '8722'
y = int(400)    # y will be 400
z = float(92)  # z will be 92.0
print(x)
print(y)

#Get the Type
print(type(x))
print(type(y))

#Single or Double Quotes
x = "Python"
# is the same as
x = 'Python'

#Case Sensitive
p = 10
P = "Python"
print(p)
print(P)
```

```
8722
Python
8722
400
<class 'str'>
<class 'int'>
10
Python
```

(i) Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Multi Words Variable Names

- Variable names with more than one word can be difficult to read.
- There are several techniques you can use to make them more readable: Camel Case,Pascal Case,Snake Case


#Legal variable names
myvar = "Python"
my_var = "Python"
_my_var = "Python"
myVar = "Python"
MYVAR = "Python"
myvar2 = "Python"
#Camel Case
myVariableName = "Python"
#Pascal Case
MyVariableName = "Python"
#Snake Case
my_variable_name = "Python"

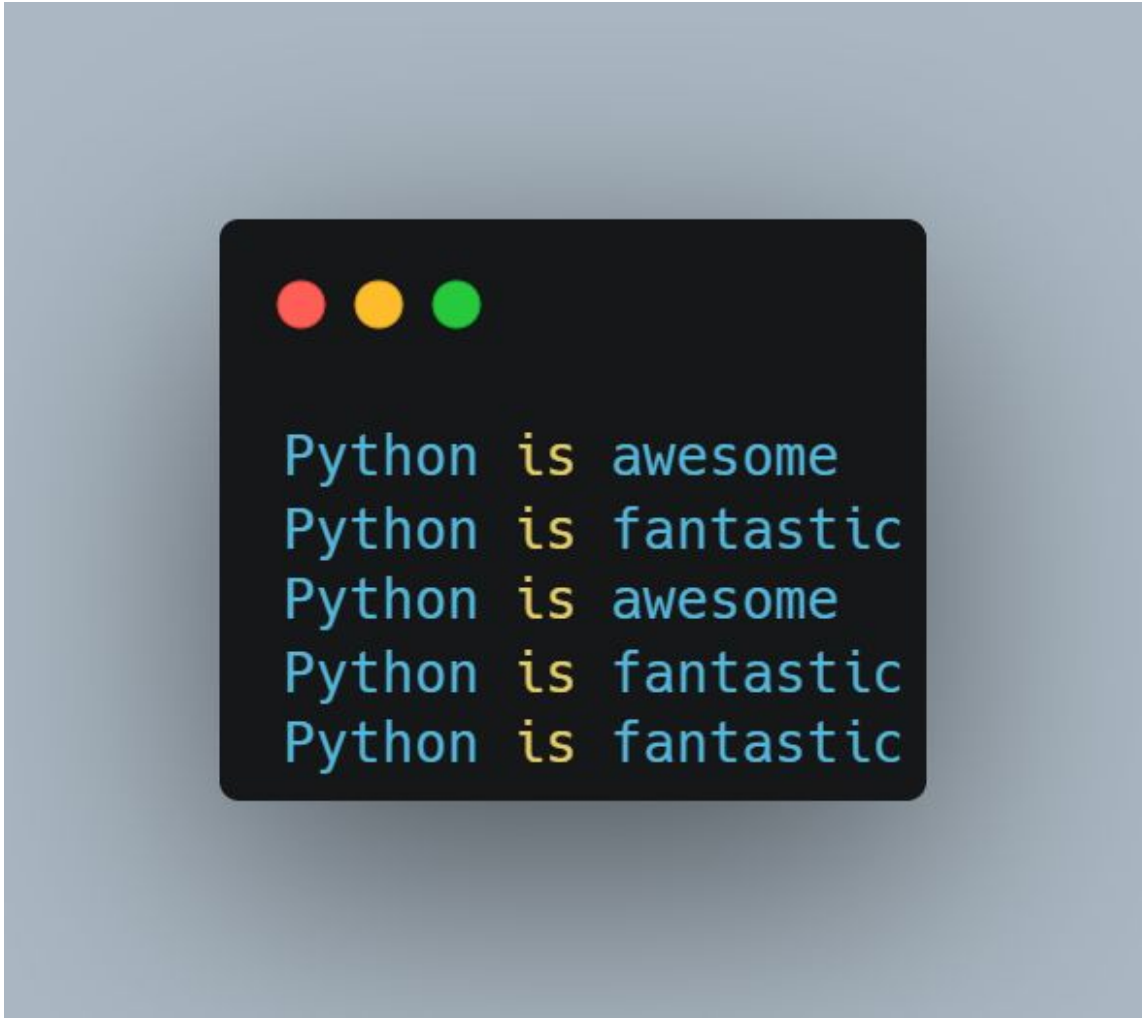
```python
#Python Variables - Assign Multiple Values
x, y, z = "Orange", "Strawberry", "Cherry"
print(x)
print(y)
print(z)

x = y = z = "Watermelon"
print(x)
print(y)
print(z)

fruits = ["apple", "Grapes", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

```
Orange
Strawberry
Cherry
Watermelon
Watermelon
Watermelon
apple
Grapes
cherry
```

```python
#Output Variables
#The Python print() function is often used to output variables.
x = "Python is awesome"
print(x)

x = "Python"
y = "is"
z = "awesome"
print(x, y, z)

x = "Python "
y = "is "
z = "awesome"
print(x + y + z)

x = 5
y = "Python"
print(x, y)
```

```
Python is awesome
Python is awesome
Python is awesome
5 Python
```

```python
# Global Variables
# Variables that are created outside of a function (as in all of the examples above) are known as global variables.

# Global variables can be used by everyone, both inside of functions and outside.
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()


x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()

print("Python is " + x)


# The global Keyword
# Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

# To create a global variable inside a function, you can use the global keyword.

def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)

x = "awesome"


def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

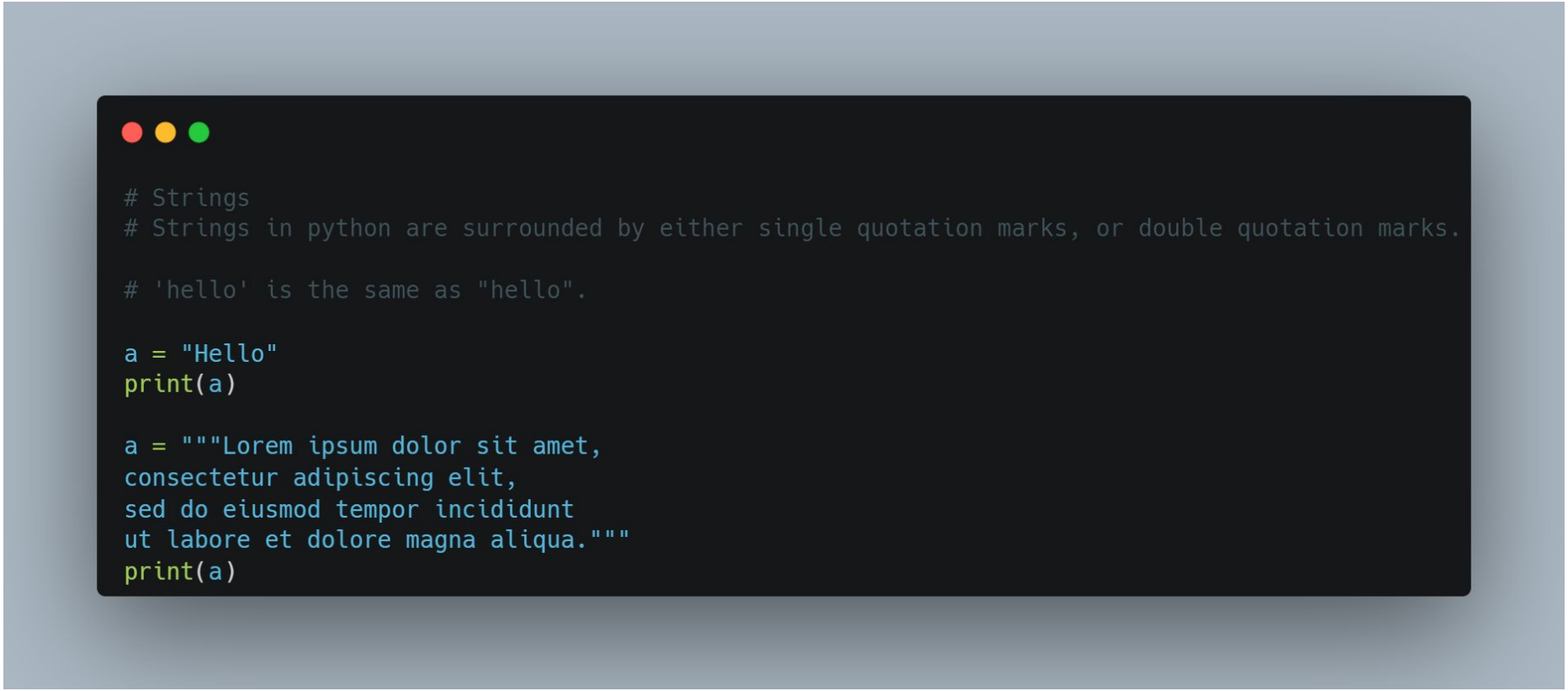

Python is awesome
Python is fantastic
Python is awesome
Python is fantastic
Python is fantastic

```
Python Data Types

Built-in Data Types
In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type:  str
Numeric Types:  int, float, complex
Sequence Types: list, tuple, range
Mapping Type:   dict
Set Types:  set, frozenset
Boolean Type:   bool
Binary Types:   bytes, bytearray, memoryview
None Type:  NoneType
```

```python
# Python Numbers

# There are three numeric types in Python:

# int
# float
# complex
# Variables of numeric types are created when you assign a value to them

x = 1    # int
y = 2.8  # float
z = 1j   # complex

# Random Number
# Python does not have a random() function to make a random number, but Python has a built-in module
called random that can be used to make random numbers:
import random

print(random.randrange(1, 10))
```

```
5
```

```
# Python Casting
Specify a Variable Type
There may be times when you want to specify a type on to a variable. This can be done with casting.
Python is an object-orientated language, and as such it uses classes to define data types, including its
primitive types.

Casting in python is therefore done using constructor functions:

int() - constructs an integer number from an integer literal, a float literal (by removing all decimals),
or a string literal (providing the string represents a whole number)
float() - constructs a float number from an integer literal, a float literal or a string literal
(providing the string represents a float or an integer)
str() - constructs a string from a wide variety of data types, including strings, integer literals and
float literals

x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
```



```python
# Strings
# Strings in python are surrounded by either single quotation marks, or double quotation marks.

# 'hello' is the same as "hello".

a = "Hello"
print(a)

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

```
Hello
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
```

```python
# Slicing
# You can return a range of characters by using the slice syntax.

# Specify the start index and the end index, separated by a colon, to return a part of the string.
b = "Hello, World!"
print(b[:5])
```

Hello

```python
# Python - Modify Strings
# Python has a set of built-in methods that you can use on strings.
a = "Hello, World!"
print(a.upper())
print(a.lower())
print(a.strip()) # returns "Hello, World!"
print(a.replace("H", "J"))
print(a.split(",")) # returns ['Hello', ' World!']
```

HELLO, WORLD!
hello, world!
Hello, World!
Jello, World!
['Hello', ' World!']

```python
# String Concatenation
# To concatenate, or combine, two strings you can use the + operator.
a = "Hello"
b = "World"
c = a + b
print(c)
c = a + " " + b
print(c)
```

```
HelloWorld
Hello World
```

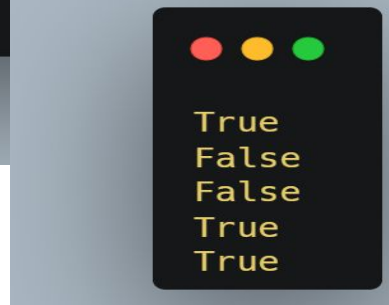
```python
# String Format
# we cannot combine strings and numbers directly.
# But we can combine strings and numbers by using the format() method!

# The format() method takes the passed arguments, formats them, and places them in the string where the
placeholders {} are:
age = 20
txt = "My name is AI, and I am {}"
print(txt.format(age))

quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))

myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

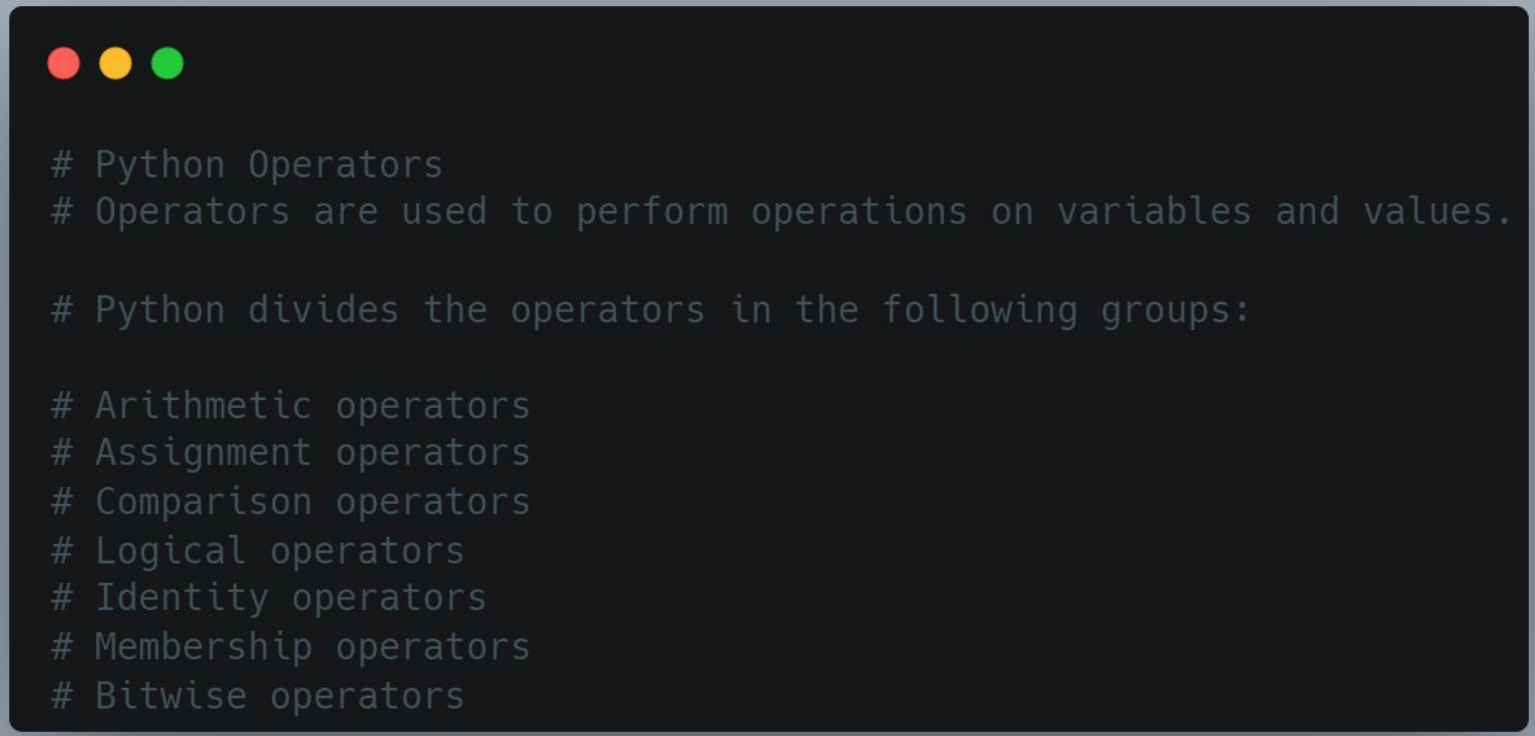
```
My name is AI, and I am 20
I want 3 pieces of item 567 for 49.95 dollars.
I want to pay 49.95 dollars for 3 pieces of
item 567.
```

```python
# Python - Escape Characters
# Escape Character
# To insert characters that are illegal in a string, use an escape character.

# An escape character is a backslash \ followed by the character you want to insert.
txt = "We are the so-called \"Vikings\" from the north."
```

```python
# String Methods
# Python has a set of built-in methods that you can use on strings.
# capitalize(),casefold(),center(),count(),find(),etc..
```

```python
# Python Booleans
# Boolean Values
# In programming you often need to know if an expression is True or False.

# You can evaluate any expression in Python, and get one of two answers, True or False.

# When you compare two values, the expression is evaluated and Python returns the Boolean answer:

print(10 > 9)
print(10 == 9)
print(10 < 9)
print(bool("Hello"))
print(bool(15))
```




Python Operators
Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

Arithmetic operators
Assignment operators
Comparison operators
Logical operators
Identity operators
Membership operators
Bitwise operators

Python Lists
Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Lists are created using square brackets

List items are ordered, changeable, and allow duplicate values.

List items are indexed, the first item has index [0], the second item has index [1] etc.

Allow Duplicates

List Length

To determine how many items a list has, use the len() function

List items can be of any data type

A list can contain different data types

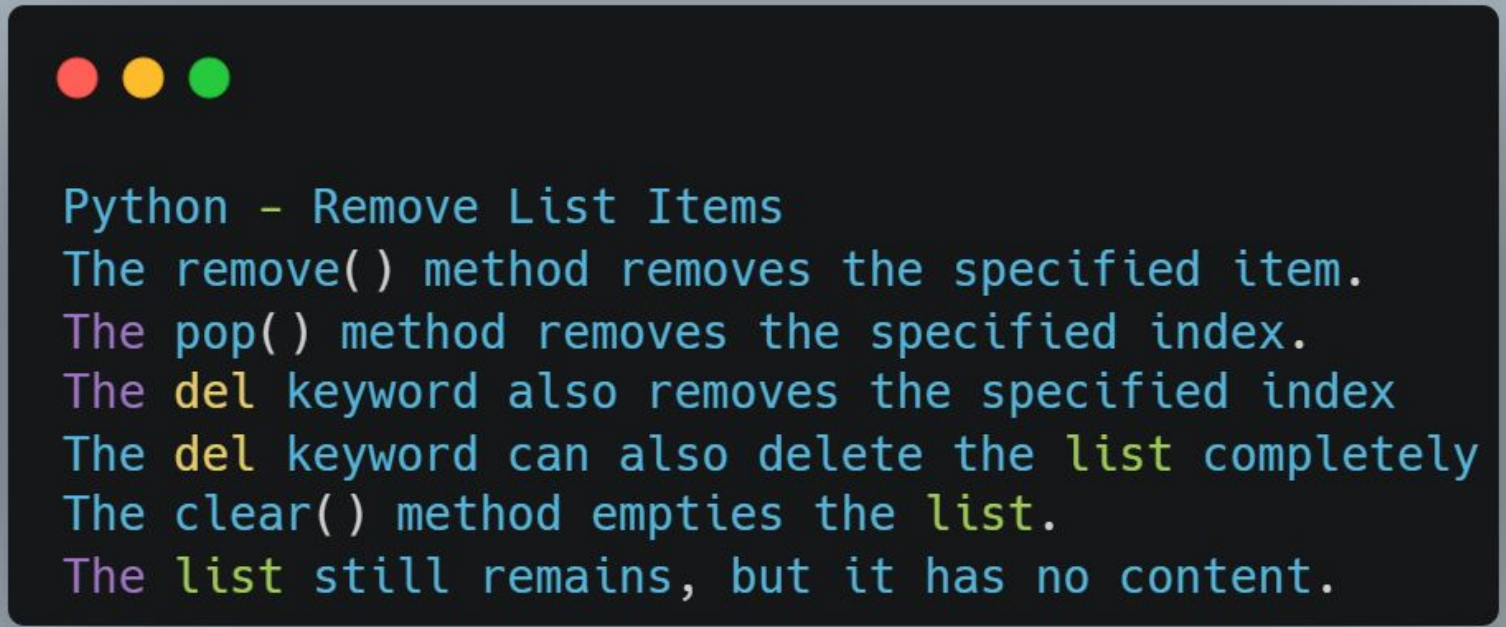
It is also possible to use the list() constructor when creating a new list.

Examples:

```
thislist = list(("apple", "banana", "cherry")) # note the double round-brackets

list1 = ["abc", 34, True, 40, "male"]
```

```
Append Items
To add an item to the end of the list, use the append() method

thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)



Insert Items
To insert a list item at a specified index, use the insert() method.
The insert() method inserts an item at the specified index

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


Extend List
To append elements from another list to the current list, use the extend() method.

thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)



Add Any Iterable
The extend() method does not have to append lists, you can add any iterable object (tuples, sets,
dictionaries etc.).

thislist = ["apple", "banana", "cherry"]
thistuple = ("kiwi", "orange")
thislist.extend(thistuple)
print(thislist)
```

```
['apple', 'banana', 'cherry', 'orange']
['apple', 'orange', 'banana', 'cherry']
['apple', 'banana', 'cherry', 'mango',
'pineapple', 'papaya']
['apple', 'banana', 'cherry', 'kiwi', 'orange']
```


Python - Remove List Items
The remove() method removes the specified item.
The pop() method removes the specified index.
The del keyword also removes the specified index
The del keyword can also delete the list completely
The clear() method empties the list.
The list still remains, but it has no content.

```python
# Python - List Comprehension
# List Comprehension
# List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

# Example:

# Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.

# Without list comprehension you will have to write a for statement with a conditional test inside

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

['apple', 'banana', 'mango']

```python
# Customize Sort Function
# You can also customize your own function by using the keyword argument key = function.
# The function will return a number that will be used to sort the list (the lowest number first)

def myfunc(n):
  return abs(n - 50)

thislist = [100, 50, 65, 82, 23]
thislist.sort(key = myfunc)
print(thislist)

# Case Insensitive Sort
# By default the sort() method is case sensitive, resulting in all capital letters being sorted before
lower case letters

thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort()
print(thislist)

thislist.sort(key = str.lower)
print(thislist)

# Reverse Order
# What if you want to reverse the order of a list, regardless of the alphabet
# The reverse() method reverses the current sorting order of the elements.

thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist)
```

```python
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist)
```

```
# Python Tuples
Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List,
Set, and Dictionary, all with different qualities and usage.

A tuple is a collection which is ordered and unchangeable.

Tuples are written with round brackets.

Tuple items are ordered, unchangeable, and allow duplicate values.

Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

mytuple = ("apple", "banana", "cherry")
```

```python
# Change Tuple Values
# Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable as it also is called.

# But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)

# Add tuple to a tuple. You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple:

thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y
print(thistuple)
```

```
('apple', 'kiwi', 'cherry')
('apple', 'banana', 'cherry', 'orange')
```

```python
# Python - Unpack Tuples

# Unpacking a Tuple
# When we create a tuple, we normally assign values to it. This is called "packing" a tuple:
fruits = ("apple", "banana", "cherry")

# But, in Python, we are also allowed to extract the values back into variables. This is called
"unpacking":

(green, yellow, red) = fruits
print(green)
print(yellow)
print(red)

# Using Asterisk*
# If the number of variables is less than the number of values, you can add an * to the variable name and
the values will be assigned to the variable as a list:

(green, yellow, *red) = fruits
print(green)
print(yellow)
print(red)
```

```
apple
banana
cherry
apple
banana
['cherry']
```

```
Tuple Methods
Python has two built-in methods that you can use on tuples.

Method  Description
count() Returns the number of times a specified value occurs in a tuple
index() Searches the tuple for a specified value and returns the position of where it was found
```

```
Python Sets
Sets are used to store multiple items in a single variable.

Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List,
Tuple, and Dictionary, all with different qualities and usage.

A set is a collection which is unordered, unchangeable*, and unindexed.

Sets are written with curly brackets.

Set items are unordered, unchangeable, and do not allow duplicate values.

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

```python
# Access Items
# You cannot access items in a set by referring to an index or a key.

# But you can loop through the set items using a for loop, or ask if a specified value is present in a
set, by using the in keyword.
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)

print("banana" in thisset)
```

```
cherry
apple
banana
True
```

```python
# Add Items
# To add one item to a set use the add() method.

thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
print(thisset)

# Add Sets
# To add items from another set into the current set, use the update() method.
tropical = {"pineapple", "mango", "papaya"}
thisset.update(tropical)
print(thisset)

# Add Any Iterable
# The object in the update() method does not have to be a set, it can be any iterable object (tuples,
lists, dictionaries etc.).
mylist = ["kiwi", "orange"]
thisset.update(mylist)
print(thisset)
```

```
{'orange', 'cherry', 'apple', 'banana'}
{'apple', 'orange', 'pineapple', 'papaya',
'mango', 'cherry', 'banana'}
{'kiwi', 'apple', 'orange', 'pineapple', 'papaya',
'mango', 'cherry', 'banana'}
```

Remove Item
To remove an item in a set, use the remove(), or the discard() method.

If the item to remove does not exist, remove() will raise an error.
If the item to remove does not exist, discard() will NOT raise an error.

You can also use the pop() method to remove an item, but this method will remove the last item. Remember that sets are unordered, so you will not know what item that gets removed.
The return value of the pop() method is the removed item.

The clear() method empties the set

The del keyword will delete the set completely

Join Two Sets
There are several ways to join two or more sets in Python.

You can use the union() method that returns a new set containing all items from both sets, or the update() method that inserts all the items from one set into another
The intersection_update() method will keep only the items that are present in both sets.
The intersection() method will return a new set, that only contains the items that are present in both sets.
The symmetric_difference_update() method will keep only the elements that are NOT present in both sets.
The symmetric_difference() method will return a new set, that contains only the elements that are NOT present in both sets.

```python
# Python Dictionaries
# Dictionaries are used to store data values in key:value pairs.
# A dictionary is a collection which is ordered*, changeable and do not allow duplicates.
# Dictionaries are written with curly brackets, and have keys and values
# Dictionary items are ordered, changeable, and does not allow duplicates.
# Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

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

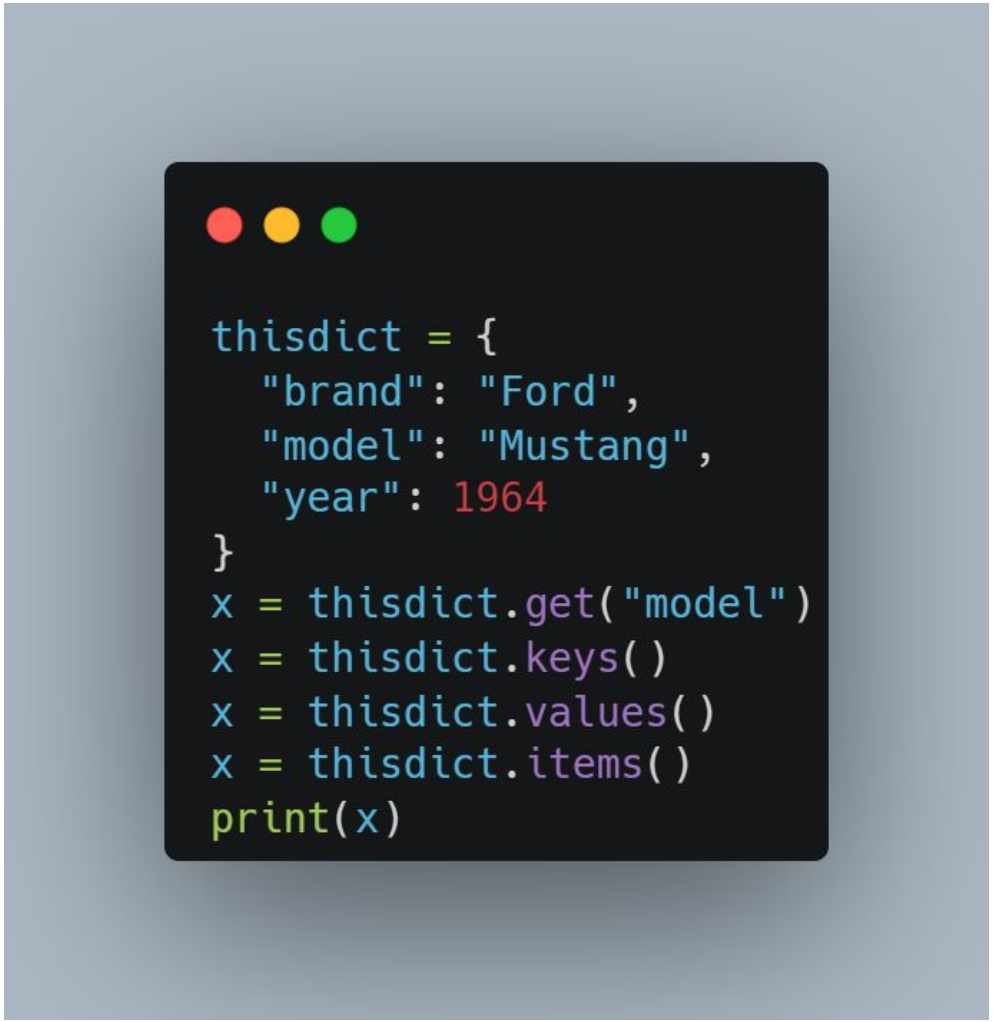
Python - Access Dictionary Items
Accessing Items

You can access the items of a dictionary by referring to its key name, inside square brackets

There is also a method called get() that will give you the same result

The keys() method will return a list of all the keys in the dictionary.
The list of the keys is a view of the dictionary, meaning that any changes done to the dictionary will be reflected in the keys list.
The values() method will return a list of all the values in the dictionary.
The list of the values is a view of the dictionary, meaning that any changes done to the dictionary will be reflected in the values list.
The items() method will return each item in a dictionary, as tuples in a list.

Check if Key Exists
To determine if a specified key is present in a dictionary use the in keyword



dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])

```python
# Update Dictionary
# The update() method will update the dictionary with the items from the given argument.

# The argument must be a dictionary, or an iterable object with key:value pairs.

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"year": 2020})
print(thisdict)
```

{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}

```
Removing Items
There are several methods to remove items from a dictionary
The pop() method removes the item with the specified key name
The popitem() method removes the last inserted item (in versions before 3.7, a random item is removed
instead)
The del keyword removes the item with the specified key name
The clear() method empties the dictionary
```

```python
# Copy a Dictionary
# You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a reference to dict1, and changes made in dict1 will automatically also be made in dict2.

# There are ways to make a copy, one way is to use the built-in Dictionary method copy().

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

# Another way to make a copy is to use the built-in function dict().
mydict = dict(thisdict)
print(mydict)
```

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

```
# Nested Dictionaries
# A dictionary can contain dictionaries, this is called nested dictionaries.

myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}

# Or, if you want to add three dictionaries into a new dictionary:

child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
```

File handling is an important part of any web application.

Python has several functions for creating, reading, updating, and deleting files.

```
# Python File Open
The key function for working with files in Python is the open() function.

The open() function takes two parameters; filename, and mode.
There are four different methods (modes) for opening a file:
"r" - Read - Default value. Opens a file for reading, 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 the file exists

In addition you can specify if the file should be handled as binary or text mode
"t" - Text - Default value. Text mode
"b" - Binary - Binary mode (e.g. images)

f = open("demofile.txt", "rt")
```

```python
#Read
# The open() function returns a file object, which has a read() method for reading the content of the
file:
f = open("demofile.txt", "r")
print(f.read())

# Return the 5 first characters of the file:

f = open("demofile.txt", "r")
print(f.readline())
print(f.read(5))
```


# Python File Write
f = open("myfile.txt", "w")

```python
# Delete a File
# To delete a file, you must import the OS module, and run its os.remove() function:

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

# Check if File exist:
# To avoid getting an error, you might want to check if the file exists before you try to delete it:

import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")

# Delete Folder
# To delete an entire folder, use the os.rmdir() method:

import os
os.rmdir("myfolder")
```