Python Introduction

Dr. Ashima

What is Python?

- 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.

Python Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.
- print("Hello, World!")

Python Getting Started

- 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):
- C:\Users\Your Name>python --version
- To check if you have python installed on a Linux or Mac, then on linux open the command line or on Mac open the Terminal and type:
- python --version
- If you find that you do not have python installed on your computer, then you can download it for free from the following website: https://www.python.org/

Python Quickstart

- Python is an interpreted programming language, this means that as a developer you write Python (.py) files in a text editor and then put those files into the python interpreter to be executed.
- The way to run a python file is like this on the command line:
- C:\Users\Your Name>python helloworld.py
- Where "helloworld.py" is the name of your python file.
- Let's write our first Python file, called helloworld.py, which can be done in any text editor.
- helloworld.py
- print("Hello, World!")

The Python Command Line

- To test a short amount of code in python sometimes it is quickest and easiest not to write the code in a file. This is made possible because Python can be run as a command line itself.
- Type the following on the Windows, Mac or Linux command line:
- C:\Users\Your Name>python
- Or, if the "python" command did not work, you can try "py":C:\Users\Your Name>py
- From there you can write any python, including our hello world example from earlier in the tutorial:
- C:\Users\Your Name>python
  Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
  Type "help", "copyright", "credits" or "license" for more information.
  >>> print("Hello, World!")

- Which will write "Hello, World!" in the command line:

-
  ```
  C:\Users\Your Name>python
  Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
  Type "help", "copyright", "credits" or "license" for more information.
  >>> print("Hello, World!")
  Hello, World!
  ```
- Whenever you are done in the python command line, you can simply type the following to quit the python command line interface:
- exit()

Python Syntax

- Execute Python Syntax
- As we learned in the previous page, Python syntax can be executed by writing directly in the Command Line:

-
  ```
  >>> print("Hello, World!")
  Hello, World!
  ```
- Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:
- C:\Users\Your Name>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.

-
  ```
  if 5 > 2:
    print("Five is greater than two!")
  ```
- Python will give you an error if you skip the indentation:
- Example
- Syntax Error:

-
  ```
  if 5 > 2:
  print("Five is greater than two!")
  ```

- The number of spaces is up to you as a programmer, but it has to be at least one.
- Example

-
  ```
  if 5 > 2:
   print("Five is greater than two!")
  if 5 > 2:
          print("Five is greater than two!")
  ```
- ou have to use the same number of spaces in the same block of code, otherwise Python will give you an error:
- Example
- Syntax Error:

-
  ```
  if 5 > 2:
   print("Five is greater than two!")
          print("Five is greater than two!")
  ```

Python Variables

- In Python, variables are created when you assign a value to it:
- Example
- Variables in Python:
- x = 5
  y = "Hello, World!“
- Python has no command for declaring a variable.
- You will learn more about variables in the Python Variables chapter.

Comments

- Python has commenting capability for the purpose of in-code documentation.
- Comments start with a #, and Python will render the rest of the line as a comment:
- Example
- Comments in Python:
- #This is a comment.
  print("Hello, World!")

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.

Creating a Comment

- Comments starts with a #, and Python will ignore them:
- Example
- #This is a comment
  print("Hello, World!“)
- Comments can be placed at the end of a line, and Python will ignore the rest of the line:
- print("Hello, World!") #This is a comment
- A comment does not have to be text that explains the code, it can also be used to prevent Python from executing code:
- #print("Hello, World!")
  print("Cheers, Mate!")

Multi Line Comments

- Python does not really have a syntax for multi line comments.
- To add a multiline comment you could insert a # for each line:

-
  ```
  #This is a comment
  #written in
  #more than just one line
  print("Hello, World!")
  ```
- Or, not quite as intended, you can use a multiline string.
- Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

-
  ```
  """
  This is a comment
  written in
  more than just one line
  """
  print("Hello, World!")
  ```

- As long as the string is not assigned to a variable, Python will read the code, but then ignore it, and you have made a multiline comment.

Python 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.

-
  ```
  x = 5
  y = "John"
  print(x)
  print(y)
  ```

- Variables do not need to be declared with any particular *type*, and can even change type after they have been set.

-
  ```
  x = 4       # x is of type int
  x = "Sally" # x is now of type str
  print(x)
  ```

Casting

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

-
  ```
  x = str(3)    # x will be '3'
  y = int(3)    # y will be 3
  z = float(3)  # z will be 3.0

  print(x)

  print(y)

  print(z)
  ```

Get the Type

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

-
  ```
  x = 5
  y = "John"
  print(type(x))
  print(type(y))
  ```

Single or Double Quotes?

- String variables can be declared either by using single or double quotes:
- x = "John"
  # is the same as
  x = 'John'

Case-Sensitive

- Variable names are case-sensitive.
- a = 4

  A = "Sally"

  print(a)

  print(A)

This will create two variables: # A will not overwrite a

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)

Example

- myvar = "John"
- my_var = "John"
- _my_var = "John"
- myVar = "John"
- MYVAR = "John"
- myvar2 = "John"

- print(myvar)
- print(my_var)
- print(_my_var)
- print(myVar)
- print(MYVAR)
- print(myvar2)

Example

- Illegal variable names:
- 2myvar = "John"
- my-var = "John"
- my var = "John"

- #This example will produce an error in the result
- Remember that variable names are case-sensitive

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

- Each word, except the first, starts with a capital letter:
- myVariableName = "John“

Pascal Case

- Each word starts with a capital letter:
- MyVariableName = "John"

Snake Case

- Each word is separated by an underscore character:
- my_variable_name = "John"

Assign Multiple Values

Many Values to Multiple Variables

Python allows you to assign values to multiple variables in one line:

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

Make sure the number of variables matches the number of values, or else you will get an error.

- **One Value to Multiple Variables**
- And you can assign the *same* value to multiple variables in one line:

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

- **Unpack a Collection**
- If you have a collection of values in a list, tuple etc. Python allows you extract the values into variables. This is called *unpacking*.

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

Output Variables

- The python print statement is often used to output variables.
- To combine both text and a variable, python uses the + character:
- x = "awesome"
- print("Python is " + x)
- You can also use the + character to add a variable to another variable:
- x = "Python is "
- y = "awesome"
- z = x + y
- print(z)

- For numbers, the + character works as a mathematical operator:

```
x = 5
y = 10
print(x + y)
```

If you try to combine a string and a number, Python will give you an error:

```
x = 5
y = "John"
print(x + y)
```

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

Getting the Data Type

- You can get the data type of any object by using the type() function:
- x = 5
- print(type(x))
- Setting the Data Type
- In Python, the data type is set when you assign a value to a variable:

Example	Data Type	
x = "Hello World"	str	
x = 20	int	
x = 20.5	float	
x = 1j	complex	
x = ["apple", "banana", "cherry"]	list	
x = ("apple", "banana", "cherry")	tuple	
x = range(6)	range	
x = {"name" : "John", "age" : 36}	dict	
x = {"apple", "banana", "cherry"}	set	
x = frozenset({"apple", "banana", "cherry"})	frozenset	
x = True	bool	
x = b"Hello"	bytes	
x = bytearray(5)	bytearray	
x = memoryview(bytes(5))	memoryview	

Setting the Specific Data Type

- If you want to specify the data type, you can use the following constructor functions:

Example	Data Type
`x = str("Hello World")`	str
`x = int(20)`	int
`x = float(20.5)`	float
`x = complex(1j)`	complex
`x = list(("apple", "banana", "cherry"))`	list
`x = tuple(("apple", "banana", "cherry"))`	tuple
`x = range(6)`	range
`x = dict(name="John", age=36)`	dict
`x = set(("apple", "banana", "cherry"))`	set
`x = frozenset(("apple", "banana", "cherry"))`	frozenset
`x = bool(5)`	bool
`x = bytes(5)`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview

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:

Example

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

Examples

-
  ```
  x = 1
  y = 35656222554887711
  z = -3255522
  print(type(x))
  print(type(y))
  print(type(z))
  ```

-
  ```
  x = 1.10
  y = 1.0
  z = -35.59

  print(type(x))
  print(type(y))
  print(type(z))
  ```

-
  ```
  x = 35e3
  y = 12E4
  z = -87.7e100

  print(type(x))
  print(type(y))
  print(type(z))
  ```

-
  ```
  x = 3+5j
  y = 5j
  z = -5j

  print(type(x))
  print(type(y))
  print(type(z))
  ```

Type Conversion

- you can convert from one type to another with int(), float(), and complex () methods:
- Example:
-
```
x = 1    # int
y = 2.8  # float
z = 1j   # complex

#convert from int to float:
a = float(x)

#convert from float to int:
b = int(y)

#convert from int to complex:
c = complex(x)

print(a)
print(b)
print(c)

print(type(a))
print(type(b))
print(type(c))
```
- You cannot convert complex numbers into another number type

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:

- Example
- Import the random module, and display a random number between 1 and 9:
- Example
- import random

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

Python Strings

Strings

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

'hello' is the same as "hello".

You can display a string literal with the print() function:

Example

```
print("Hello")
print('Hello')
```

Assign String to a Variable

- Assigning a string to a variable is done with the variable name followed by an equal sign and the string:
- Example
- a = "Hello"
  print(a)

Multiline Strings

- You can assign a multiline string to a variable by using three quotes:
- Example
- You can use three double quotes:
- a = """Lorem ipsum dolor sit amet,
  consectetur adipiscing elit,
  sed do eiusmod tempor incididunt
  ut labore et dolore magna aliqua."""
  print(a)

- Or three single quotes:
- Example
- a = '''Lorem ipsum dolor sit amet,
  consectetur adipiscing elit,
  sed do eiusmod tempor incididunt
  ut labore et dolore magna aliqua.'''
  print(a)

Strings are Arrays

- Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.
- However, Python does not have a character data type, a single character is simply a string with a length of 1.
- Square brackets can be used to access elements of the string.
- Example
- Get the character at position 1 (remember that the first character has the position 0):
- a = "Hello, World!"
  print(a[1])

Looping Through a String

- Since strings are arrays, we can loop through the characters in a string, with a for loop.

- Example
- Loop through the letters in the word "banana":

- for x in "banana":
-   print(x)

String Length

- To get the length of a string, use the len() function.

- Example
- The len() function returns the length of a string:

- a = "Hello, World!"
- print(len(a))

Check String

- To check if a certain phrase or character is present in a string, we can use the keyword in.

- Example
- Check if "free" is present in the following text:

- txt = "The best things in life are free!"
- print("free" in txt)

- Use it in an if statement:

- Example
- Print only if "free" is present:

- txt = "The best things in life are free!"
- if "free" in txt:
-   print("Yes, 'free' is present.")

Check if NOT

- To check if a certain phrase or character is NOT present in a string, we can use the keyword not in.

- Example
- Check if "expensive" is NOT present in the following text:

- txt = "The best things in life are free!"
- print("expensive" not in txt)

- Use it in an if statement:

- Example
- print only if "expensive" is NOT present:

- txt = "The best things in life are free!"
- if "expensive" not in txt:
-   print("No, 'expensive' is NOT present.")

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.
- Example
- Get the characters from position 2 to position 5 (not included):
- b = "Hello, World!"
  print(b[2:5])
-

Slice From the Start

- By leaving out the start index, the range will start at the first character:
- Example
- Get the characters from the start to position 5 (not included):
- b = "Hello, World!"
  print(b[:5])

Slice To the End

- By leaving out the *end* index, the range will go to the end:
- Example
- Get the characters from position 2, and all the way to the end:
- b = "Hello, World!"
  print(b[2:])

Negative Indexing

- Use negative indexes to start the slice from the end of the string:Example
- Get the characters:
- From: "o" in "World!" (position -5)
- To, but not included: "d" in "World!" (position -2):
- b = "Hello, World!"
  print(b[-5:-2])

Python - Modify Strings

- Python has a set of built-in methods that you can use on strings.

- Upper Case
- Example
- The upper() method returns the string in upper case:

- a = "Hello, World!"
- print(a.upper())

- Lower Case
- Example
- The lower() method returns the string in lower case:
- a = "Hello, World!"
- print(a.lower())
- Remove Whitespace
- Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

- Example
- The strip() method removes any whitespace from the beginning or the end:
- a = " Hello, World! "
- print(a.strip()) # returns "Hello, World!"

- Split String
- The split() method returns a list where the text between the specified separator becomes the list items.

- Example
- The split() method splits the string into substrings if it finds instances of the separator:

- a = "Hello, World!"
- print(a.split(",")) # returns ['Hello', ' World!']

- String Concatenation
- To concatenate, or combine, two strings you can use the + operator.

- Example
- Merge variable a with variable b into variable c:

- a = "Hello"
- b = "World"
- c = a + b
- print(c)

- Example
- To add a space between them, add a " ":

- a = "Hello"
- b = "World"
- c = a + " " + b
- print(c)

- String Format
- As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:
- Example
- age = 36
  txt = "My name is John, I am " + age
  print(txt)
- 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:

- Example
- Use the format() method to insert numbers into strings:

- age = 36
- txt = "My name is John, and I am {}"
- print(txt.format(age))

- The format() method takes unlimited number of arguments, and are placed into the respective placeholders:
- Example
- quantity = 3
  itemno = 567
  price = 49.95
  myorder = "I want {} pieces of item {} for {} dollars."
  print(myorder.format(quantity, itemno, price))

- You can use index numbers {0} to be sure the arguments are placed in the correct placeholders:

- Example
- quantity = 3
- itemno = 567
- price = 49.95
- myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
- print(myorder.format(quantity, itemno, price))

String Methods

- Python has a set of built-in methods that you can use on strings.
- **Note:** All string methods returns new values. They do not change the original string.

Python Booleans

- Booleans represent one of two values: True or False.

- 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:

- Example
- print(10 > 9)
- print(10 == 9)
- print(10 < 9)

- When you run a condition in an if statement, Python returns True or False:

- Example
- Print a message based on whether the condition is True or False:

- a = 200
- b = 33

- if b > a:
-   print("b is greater than a")
- else:
-   print("b is not greater than a")

Evaluate Values and Variables

- The bool() function allows you to evaluate any value, and give you True or False in return,

- Example
- Evaluate a string and a number:

- print(bool("Hello"))
- print(bool(15))

- Example
- Evaluate two variables:

-
  ```
  x = "Hello"
  y = 15

  print(bool(x))
  print(bool(y))
  ```

Most Values are True

- Almost any value is evaluated to True if it has some sort of content.

- Any string is True, except empty strings.

- Any number is True, except 0.

- Any list, tuple, set, and dictionary are True, except empty ones.

- Example
- The following will return True:

- bool("abc")
- bool(123)
- bool(["apple", "cherry", "banana"])

Some Values are False

- In fact, there are not many values that evaluate to False, except empty values, such as (), [], {}, "", the number 0, and the value None. And of course the value False evaluates to False.

- Example
- The following will return False:

- bool(False)
- bool(None)
- bool(0)
- bool("")
- bool(())
- bool([])
- bool({})

- One more value, or object in this case, evaluates to False, and that is if you have an object that is made from a class with a __len__ function that returns 0 or False:

- Example
- class myclass():
-   def __len__(self):
-     return 0

- myobj = myclass()
- print(bool(myobj))

Functions can Return a Boolean

- You can create functions that returns a Boolean Value:
- Example
- Print the answer of a function:

-
  ```
  def myFunction() :
    return True

  print(myFunction())
  ```

- You can execute code based on the Boolean answer of a function:

- Example
- Print "YES!" if the function returns True, otherwise print "NO!":

- def myFunction() :
-   return True

- if myFunction():
-   print("YES!")
- else:
-   print("NO!")

- Python also has many built-in functions that return a boolean value, like the isinstance() function, which can be used to determine if an object is of a certain data type:

- Example
- Check if an object is an integer or not:

- x = 200
- print(isinstance(x, int))

Python Operators

- Operators are used to perform operations on variables and values.
- In the example below, we use the + operator to add together two values:
- Example
- print(10 + 5)

- Python divides the operators in the following groups:

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

Practice Question

- Write a python program to add , subtract, multiply, divide the the two numbers by taking input values and without taking the input values?
- Write a Python program to find the average of three numbers
- Python program to find the average of three numbers by taking input values
- Write a Python program to calculate the simple interest
- Write a Python program to calculate the compound interest
- Write a Python program to find the square root

- Write a Python program to find the area of the circle.
- Write a Python program to find the area of the rectangle.
- Write a Python program to find the area of the right-angle triangle.
- Write a Python program to swap two variables using temporary variable
- Write a python program to covert Fahrenheit to Celsius Formula
- Write a program to Display Calendar of a Month
- write a Python program to convert number of days into years, weeks and days