Python Basics

Basics of Python for Data Analysis

- Open Source – free to install
- Very easy to learn
- Can become a common language for data science and production of web based analytics products.
- It is an interpreted language and compiled language

Python Properties

1. *Strongly* typed. It enforces data types so you can’t concatenate a string and a integer, for example.

2. *Dynamically*, *implicitly* typed. So, you don’t have to explicitly declare variable data types. Data types are enforced at runtime.

3. *Case sensitive*. For example, token and TOKEN are two different variables.

4. *Object-oriented*.

Python inventor

- Guido Van Rossum inventor of Python
- General Purpose programming language
- Build anything
- Can build packages for data science

Python Framework

- **Anaconda Framework**
- Jupyter Notebook
  - Python 3.x

Operating system version

- *Mac:*
- *Linux:*
- *Windows:*

Python script

- Python Files -.ipynb
- List of Python commands
- Executing the script

Python basic syntax

```
>>> 3+4
7
>>> 4*4
16
>>> 5-2
3
>>> 8/2
4.0
>>> a=9
>>> b=2
>>> c=a+b
>>> print(c)
11
```

python calculator

- >>> print(8+2)
- print(5 + 5)
- print(5 - 5)
- # Multiplication, division, modulo, and exponentiation
- print(3 * 5)
- print(10 / 2)
- print(18 % 7)
- print(4 ** 2)

Print -String

- >>> print ('hello world')
- hello world

- Print(“hello world”)
- Hello world

Variables

- Variables are containers for data.
- The syntax to declare them is:
  - variable_name = variable_value.
  - X=25
  - Y=69
  - Z=80

Variables example…

- number1 = 50
- number2=100
- Result=number1 +number2

- First=45
- Second=5
- Third=First+Second

Variables

```
>>> var1=89
>>> print(var1)
89
>>> var2='Hello'
>>> print(var2)
Hello
>>> addvar=8+5
>>> print(addvar)
13
```

```
>>> multiplyvar=8*5
>>> print('Multiply='+multiplyvar)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
>>> print('Multiply='+ str(multiplyvar))
Multiply=40
>>>
```

Built-in Data Types

- Python has a number of built-in data types such as numbers (integers, floats, complex numbers), strings, lists, tuples, and dictionaries.
- Each of these can be manipulated using:

- Operators
- Functions

Numbers

Numbers can be integers, floating points, Booleans, or complex numbers. The former three are the most important:

•Integers are whole numbers - 1, 2, 22, 476, -99999

•Floats have decimal points - 1.0, 2.22, 22.098, 476.1, -99999.9

•Booleans represent either True or False (or 1 or 0). They represent data that can only be one thing or another.

Operators

```
>>> 2 + 3 # Addition
5
>>> num1 = 10
>>> num2 = 9.99
>>> num3 = num1 + num2
>>> num3
19.990000000000002
```

Operators

- >>> 8 - 5 # Subtraction
- 3
- >>> 2 * 6 # Multiplication
- 12
- >>> 12 / 3 # Division
- 4.0
- >>> 7 % 3 # Modulus (returns the remainder from division)
- 1
- >>> 3 ** 2 # Raise to the power 9

operators

```
>>> savings=100
>>> factor=1.10
>>> result=savings*factor**6
>>> print(result)
177.1561000000001
```

Operators..

- >>> 2 < 5
- True
- >>> 4 > 10
- False
- >>> 3 >= 3
- True >>>
- >>> 5 == 6
- False >>>
- 6 != 9 True

Variable ---specific

- Calculating Body Mass Index-BMI

  - height=1.79
  - weight=68.7
  - bmi=weight/height**2
  - print(bmi)

Functions

- Python provides you with a number of built-in functions for manipulating integers. These are *always* available to you
- >>> float(9)
- 9.0

- \>>>int(5.7)
- 5

- \>>>int(3.45)
- 3

Strings

- >>> simple_string = "hey!"
- >>> simple_string
- 'hey!'
- >>> "hello world!"
- 'hello world!'
- >>> escaped = 'can\'t‘
- >>> escaped
- "can't"

Manipulating strings

- *Operators*
- Like numbers, you can concatenate strings (string concatenation):

- >>> “happy”+” “ +”birthday”
- ‘happy birthday’

- >>> “Jonas” + “Brother”
- JonasBrother

Functions

- len() - given a string, this function returns the length of it.

- >>> city =“London”
- >>> len(city)

Slice()

- slice() - given a start and stop value, you can access a set of, or single, character(s)

```
>>> ("Hello"[2])
 l
 >>> ("Hello"[3])
l
>>> ("Hello"[0])
H
>>> ("Hello"[0:2])
 He
```

String.format()

- Easily format values into strings

```
 >>> name=“ Maria Joe”
>>>greeting =“My name is  {} “.format(name)
>>> greeting
```

Guess the type-- gives the data type

```
>>> a=10
>>> type(a)
<class 'int'>
>>> str="hello"
>>> type(str)
<class 'str'>
>>> d=4.45
>>> type(d)
<class 'float'>
>>>
```

script1.ipynb----------------Python Script

- height=1.79
- weight=68.7

- bmi=weight/height**2

- print(bmi)

Python types

- >>>type(bmi)
- float
- day=5
- type(day)

LOOPS

```
>>> print ('loop')
loop
>>>
>>> ctr=1
>>> while condition < 10:
... print(condition)
  File "<stdin>", line 2
    print(condition)
        ^
IndentationError: expected an indented block
```