Getting Started with Python, Numpy and Pandas

Python is an interpretive language.
Python is a strongly-typed and dynamically-typed language.
Strongly-typed: Interpreter always “respects” the types of each variable.
Dynamically-typed: A variable is simply a value bound to a name.

Primitive Types

- integer
- float
- bool
- string

```
A = 5
B = 5.2
C = "True"
D = "A String"
E = '''
A
multi-line
String
'''
print (A, B, C, D, E)
```

Dynamically Typed

```
x = 5
print ("type of variable x:",type(x))
print("x = ",x)

x = "BITS Pilani"
print("type of variable x:",type(x))
print("x = ",x)

x = 5.2
print("type of variable x:",type(x))
print("x = ",x)
```

Strongly Typed

```
a = 5.2
x = "A String "
a * x
```

```
b = 5
```

```
b+y
```

String Facts: Addition of strings

```
"BITS-Pilani "+"Hyderabad Campus" # Conctenation
```

String Facts: product of string with integer

```
b*y
```

String Facts: f-string (formatted string). Allows expression evaluation inside the string.

```
z = f'Value of b*y: {b*y}'
print (z)
```

String Facts: r-string (raw string). Allows expression with escape sequences

```
z = 'The course name:\tCS F441'
print(z)
z = r'The course name:\tCS F441'
print(z)
```

Booleans and Boolean Factos

```
a = True
print(a)
b = False
print(b)
```

True is 1 and False is 0 in float/integer arithmetic and when multiplied with String

```
True*100
```

```
False*100.2
```

```
True*"String"
```

```
False*"String"
```

Comments and Multiline Comments

```
# A comment
# There is no multiline commentin Python
# But Triple quotes may be used for multi-line comments.

"""
```

```
a = True
print(a)
b = False
print(b)
"""
```

Type casting

- int()
- float()
- str()
- bool()

```
print(int(5.2))
print(int("5"))
print(float(5))
print(float("5.2"))
print(bool(0))
print(bool(5.2))
print(bool("A string"))
print(bool(""))
```

Collection Types

- Lists
- Tuple
- Dictionary
- Set

Lists are mutable (changeable) arrays

Ex: names = [‘Zach’, ‘Jay’] # note the square bracket.
Indexing to access of individual

```
names = ['Jack', 'Jill']
print(names[0])

print(print (names))

print(len(names))

names[1] = "John"
print(names)

emptyList = [] #or list()
print(len(emptyList))
```

Extend list

```
names = ['Jack', 'Jill']
names.append('Rick')
len(names)
```

```
print (names)

names.extend(['Kevin', 'Adrian'])
print (names)
```

```
eNames = ['Jack', 'Jill'] + ['Kevin', 'Adrian']
print(eNames)
```

Tuples are an ordered collection, immutable (unchangeable)

Element access is by indexing, like in list.

```
coordinates = (2., 5., 1.)     # Note the parenthesis or round brackets.

print(coordinates[0], coordinates[1])
print(len(coordinates))
```

```
emptyTuple = () #tuple()
print(emptyTuple)
oneElemTuple = (1.5, )   #Comma matters!
print(oneElemTuple)
```

```
coordinates[0] = 3
```

Dictionary: an unordered list with key-value pairs.

Elements access is using key as the index.

```
course = {
    "number" :  "CS F441",
    "name": "Data Visualization",
    "classSize": 62
}
print (course)
print (course["number"])
```

Dictionary Methods

- keys()
- values()
- items()

```
print(course.keys())

print(course.values())

print(course.items())
```

Sets: An unordered, and unindexed collection.

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

```
print(thisset)

# No duplicate items: Set items are unique
thisset = {"apple", "banana", "cherry", "apple"}
print(thisset)
```

Set Membership

```
print("banana" in thisset)

print("pineapple" in thisset)

thisset.add("pineapple")

print("pineapple" in thisset)

#Allows union, intersection,difference, ...
```

Additional Collections via External Packages

- numpy
- pandas

Note: the packages are not natively available with python. Must be installed independently prior to importing.

pip install numpy
pip install pandas

They must be imported into your code

- import numpy
- import pandas

Sometimes the package names may shortened for convenience of typing

- import numpy as np
- import pandas as pd

numpy: a Python package used for working with arrays.

Difference between list and numpy.array:

- numpy.array are non-extendable
- elements are of the same type
- Occupies continuous memory
- efficient

```
import numpy as np
A = [1, 5, 3, 4, 5]
nA = np.array(A)
print(nA)
print(nA[0])
```

```
A.append(10)
print(A)
```

```
nA.append(10)
```

```
A+5
```

```
print(nA+5)
```

```
print(nA)
```

Numpy offers random module to work with numbers

```
from numpy import random
print(random.randint(100)) # integer random number in the range [0, 100)

print(random.randint(100, size=(5))) # integer random 1D array of size 5

print (random.rand())   # float random number in the range [0, 1)

print(random.rand(5))   # float random array of size 5
```

```
print(random.uniform(low=0,high=2,size=5)) # uniform distribution in [0,2)

print(random.normal(loc=0,scale = 1, size = 5)) # normal distribution  loc is the mean
and scale is the standard deviation

print( random.choice([3, 5, 7, 9], p=[0.1, 0.3, 0.6, 0.0], size=5))

print(random.permutation([1,2,3,4,5]))
```

```
random.uniform()
```

Pandas: a Python library used for working with data sets.

Pandas stands for Panel data (tabular data).
It has functions for analyzing, cleaning, exploring, and manipulating data.
It is used for a wide range of data analysis tasks, such as data cleaning, transformation, exploration, and more.

Pandas provides data structures called DataFrame (a two-dimensional table-like data structure, like 2D array, or a table with rows and columns), and Series (a one-dimensional labeled array) that make it easier to work with and analyze data in Python.

```
import pandas as pd

student1 = {
 "courses": ["F441", "F311"],
 "grades": ["A", "B+"]
}
```

```
df = pd.DataFrame(student1)
```

```
print(student1)
```

```
print(df)
```

```
df.info()
```

```
df.columns
```

```
df.index
```

```
df.index = ["I","II"]
```

```
print(df)
```

Accessing Columns and Rows of DataFrame.

```
#column Access is similar to disctional access using key.
print(df["courses"])
```

```
print(df["grades"])
```

Pandas dataframe has two different function to access rows.

- .loc is typically used for label indexing and can access multiple columns using column name,
- .iloc is used for integer indexing both rows and columns.

```
# Row access
print(df.loc["I"])
```

```
print(df.loc["I","courses"])
```

```
print(df.iloc[0,1])
```

```
print(df.iloc[0]["courses"])
```

Reading tabular data

In pandas, you can use the read_* functions to read various types of data formats into a DataFrame. Here are a few commonly used functions for reading data.

- .read_csv()
- .read_excel()
- .read_json()
- ...

```
df = pd.read_csv("data.csv")
print(df.head(10))
```

```
df.info()
```

```
df.describe()
```

Flow Control

- if-then-else
- for loop
- while loop

if-then-else statements: 3 forms

- if statement
- if...else statement
- if...elif...else statement

Statements can be code blocks.
Code blocks are created using indents. Indentation is done using spaces: say 2 or 4 spaces. But should be consistent throughout the file.

In [ ]:
```
number = 0
```

```
#
# if statement
#
if number > 0:
    print('Number is positive.')

#
# if...else statement
#
if number > 0:
    print('Positive number')
else:
    print('Negative number')

#
# if...elif...else statement
```

```
#
if number > 0:
    print("Positive number")
elif number == 0:
    print('Zero')
else:
    print('Negative number')
```

for loop is used to run a block of code for a certain number of times.

It is used to iterate over any sequences such as range, list, tuple, string, etc.
Syntax:
for val in sequence:
# statement(s)

```
for i in range(5):
    print (i*i)
#The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.
#syntax:
#  range(start, stop, step)
```

```
aList = []
for i in range(5):
    aList.append(i*i)
print(aList)
```

Unpacking tuples

```
aList = [(1,10), (2,20), (3,30)]
for x, y in aList:
    print(x,y)
```

List Comprehension

new_list = [expression for element in iterable]
new_list = [expression for element in iterable if condition]

```
[i*i for i in range(5)]
```

```
[i*i for i in range(5) if i % 2 == 0]
```

While loop

while loop is used to run a block of code until a certain condition is met.
The syntax of while loop is

```
while condition:
   body of while loop
```

```
i = 0
while i < 5:
    print (i)
    i += 1
```

Functions:

A function is a block of code which only runs when it is called.

Function can be called with data, known as parameters (or arguments). A function can optionally return data as result.

Syntax:

```
    def function_name(arguments):
        # function body
        return
```

```
def square(num):
    return num * num
print(square(5))
```

```
for i in [1,2,3]:
    print(f'Square of {i} = {square(i)}')
```

Lambda functions

A lambda function is a small anonymous function.
A lambda function can take any number of arguments, but can only have one expression.
syntax:

```
    lambda arguments : expression
```

```
x = lambda a : a + 10
print(x(5))
```

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```