Python Notes

Python is a dynamically-typed language which means that it assigns data type to variables automatically.

It also doesn't use semi-colons and instead uses indentation.

- **Single Line Comments**

They can be given using #.

```
#Single Line Comment
#Another Single Line Comment
```

- **Multi Line Comments**

They can be given using ''' at the start and at the end.

```
'''
Multi
Line
Comment
'''
```

Indentation

Python uses spaces to indicate blocks of code. It is very important because python doesn't use semi-colons.

Here, a while loop, an if statement and a block are all nested using tabspaces.

```
while(True):
    if 5:
        {
            print(3)
        }
```

- **Python Repl**

The python Repl is a standalone environment to test lines of python code. It is useful for quick calculations.

To enter, type "**python**" in the terminal and press Enter.

To exit, type "**exit( )**" or press **Ctrl+Z**.

```
PS F:\University\Python> python
Python 3.11.6 (tags/v3.11.6:8b6ee5b, Oct  2 2023, 14:57:12) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
PS F:\University\Python>
```

Python Package Manager

pip is the package installer for Python on Windows.

- **pip install "module/library-name"**

Downloads and Installs the module/library specified.

```
PS F:\University\Python> pip install numpy
Collecting numpy
  Obtaining dependency information for numpy from https://files.pythonhosted.org/packages/93/fd/3f826c6d15d3bdcf65b8031e4835c52b7d9c45add25efa2314b53850e1a2/numpy-1.26.0-cp311-cp311-win_amd64.whl.metadata
  Using cached numpy-1.26.0-cp311-cp311-win_amd64.whl.metadata (61 kB)
Using cached numpy-1.26.0-cp311-cp311-win_amd64.whl (15.8 MB)
Installing collected packages: numpy
```

- **pip uninstall "module/library-name"**

Deletes and Uninstalls the module/library specified.

```
PS F:\University\Python> pip uninstall numpy
Found existing installation: numpy 1.26.0
Uninstalling numpy-1.26.0:
```

Import Statement

It is used to import both built-in(internal) and downloaded(external) modules/libraries.

```
import math #Internal Module
import numpy #External Module
```

Specific submodules from the module/library can also be imported.

```
import matplotlib.pyplot
```

Importing into Variables

The Module can be imported into a variable. This makes it so you don't have to write the entire module name everytime.

```
import math as m #Internal Module
import numpy as np #External Module

import matplotlib.pyplot as mpt
```

Functions

def Keyword

It is used for function declaration and definition

```
def sumFunction(x,y):
    return x+y
```

To use the function, you have to make a function call.

```
def sumFunction(x,y):
    return x+y
print(sumFunction(3,5))
```

Output: **8**

lambda Keyword

It is used for function declaration without the del keyword and within a single line. It is very useful for inlining functions and writing in the Python Repl.

Here, a lambda function is created using the lambda keyword.

```
sumFunction = lambda x,y: x+y
print(sumFunction(3,5))
```

Output: **8**

Print( ) function

used for output.

```
print("Hello World!")
```

Output: **Hello World!**

You can directly print an iterable datatype (List, Tuple, Set, Dict etc.) using print( ).

```
myList = [1,2,3,4,5]
print(myList)
```

Output: **[1,2,3,4,5]**

Input( ) function

used for input.

```
x = input("Enter your name: ")
print("Hello "+x)
```

Input: **Jaish**
Output: **Hello Jaish**

any( ) and all( ) Function

any function takes multiple arguments and returns a bool, which is only False if all arguments are False, otherwise it's True.

all function It takes multiple arguments and returns a bool, which is only True if all arguments are True, otherwise it's False.

NoneType, 0 (in any datatype) and Empty Iterables are all False. Everything else is True. They are like the OR and AND operation.

```
x = [0,0,0,1,2]
print(any(x))
print(all(x))
```

Output: **True**, **False**

Variables

Declaration and Initialization: a = 5, a = 5.5, a = [1,2,3] etc.

Rules for Variable Names: You should separate variablenames using _

- Can only start with a letter or an underscore.
- Spaces are not allowed.
- Keywords that are not allowed → and, del, from, not, while, as, elif, global, or, with, assert, else, if, pass, yield, break, except, import, print, class, exec, in, raise, continue, finally, is, return, def, for, lambda, try.

Data Types

Data Types in python are classes and Variables are objects of these classes. These are the built-in data types.

Int

A numeric data type that holds integers of non-limited length.

Float

A numeric data type that holds decimal numbers upto 15 digits of accuracy.

Complex

A numeric data type that holds complex numbers.

```
x = 10 #integer
x = 10.5 #float
x = 1+10j #complex
```

Bool

Can only be True or False.

```
x = True
y = False
```

Arithmetic Operations

1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Exponentiation (**)
5. Division (/) returns a **floating point number**.
6. Integer Division (//) returns an **integer** and ignores the **fractional part**.

```
print(10+20)
print(1-2)
print(2*3)
print(2**3)
print(1/2)
print(1//2)
```

Output: **30**, **-1**, **6**, **8**, **0.5**, **0**

Comparison

There are six comparison operators and all of them return bool (True or False).

1. Equal to (==)
2. Not Equal to (!=)
3. Less Than (<)
4. Greater Than (>)
5. Less Than or Equal to (<=)
6. Greater Than or Equal to (>=)

```
print(3==5)
print(3!=5)
print(3<5)
print(3>5)
print(3<=5)
print(3>=5)
```

Output: **False**, **True**, **True**, **False**, **True**, **False**

Iterable Data Types

Data Types which can return one element of theirs at a time. This includes Str, List, Tuple, Set and Dict.

Some are mutable while others are immutable.
Mutable = Changeable
Immutable = Unchangeable

len( ) function

It is used to get the length/size of the iterable datatype.

```
myList = [1,2,3,4,5]
print(len(myList))
```

Output: **5**

sorted( ) function

It is used to get the sorted list using some parameters.

```
myList = [3,2,5,6,7]
x = sorted(myList)
print(x)
```

Output: **[2, 3, 5, 6, 7]**

The **Reverse** parameter is used to go from Ascending order to Descending order.

```
myList = [3,2,5,6,7]
x = sorted(myList, reverse=True)
print(x)
```

Output: **[7, 6, 5, 3, 2]**

The **Key** parameter is used to add extra conditions. The Abs value takes the absolute value of each element before sorting.

```
myList = [-1,2,0,-3,5,6,-7,-4]
x = sorted(myList, key=abs)
print(x)
```

Output: **[0, -1, 2, -3, -4, 5, 6, -7]**

in Keyword

It is used to check whether an item is present in the list, tuple, set or dict. It returns bool (True or False).

```
myList = [1,2,3,4,5]
print(5 in myList)
```

Output: **True**

```
myList = [1,2,3,4,5]
print(6 in myList)
```

Output: **False**

Str

The string data type that can hold a single character or a sequence of many characters.

Operations

1. Concatenation (+) **joins** two strings.

   ```
   fName = "Jaish"
   lName = "Khan"
   name = fName + "    " + lName
   print(name)
   ```

   Output: **Jaish Khan**

2. Repetition (*) **repeats** a string multiple times.

   ```
   name = "Jaish "
   print(name*10)
   ```

   Output: **Jaish Jaish Jaish Jaish Jaish Jaish Jaish Jaish Jaish Jaish**

3. Slicing ([:]) **cuts** pieces out of the string and makes a new string.

   ```
   name = "Jaish "
   print(name[0]) #Returns the first character
   print(name[1]) #Returns the second character
   ```

   Output: **J**, **a**

   You can give **two values** (a **Starting Index** which is included and a **Stopping Index** which is excluded) separated by a **colon**.

   ```
   name = "Muhammad Jaish Khan"
   ```

```
print(name[9:14])
```
Output: **Jaish**

Negative Indexes go from last to first.
```
name = "Muhammad Jaish Khan"
print(name[-1])
```
Output: **n**

```
name = "Muhammad Jaish Khan"
print(name[-4:])
```
Output: **Khan**

List

A sequence data type that is changeable, ordered and allows duplicates.

```
list1 = [1,2,3,4,5]
```

Multi-Dimensional Lists

A list of lists is possible and so is a list of lists of lists. These are called Multi-Dimensional Lists.

A 2-Dimensional List
```
list_of_lists = [[1,2],[2,3,4],[3,4,5,6]]
```

A 3-Dimensional List
```
list_of_lists_of_lists = [[[1],[2]],[[2,3],[4,5]],[[3,4,5],[6]]]
```

List Functions

All of these change the original list.

.extend( )
It attaches individual items from the given iterable to the list.
```
myList = [3,2,5,6,7]
myList.extend([1,2])
print(myList)
```
Output: **[3, 2, 5, 6, 7, 1, 2]**

.append( )

It attaches the given iterable, as it is, to the list.

```
myList = [3,2,5,6,7]
myList.append([1,2])
print(myList)
```

Output: **[3, 2, 5, 6, 7, [1, 2]]**

.sort( )

It sorts the list and changes the list.

```
myList = [3,2,5,6,7]
myList.sort()
print(myList)
```

Output: **[2, 3, 5, 6, 7]**

It is different from the sorted( ) function.

- .sort( ) function changes the original list.
- sorted( ) function doesn't change the original list and instead makes a new one.

Tuple

A sequence data type that is unchangeable, ordered and allows duplicates.

```
tuple1 = (1,2,3,4,5)
```

It can also be written without the round brackets.

```
tuple1 = 1,2,3,4,5
```

It can be used for assigning values to multiple variables at once and swapping two variables.

```
x,y = 1,2 #Assign
print(x,y)

x,y = y,x #Swap
print(x,y)
```

Output:

1 2

2 1

Set

A set is a collection of objects that is changeable, unordered and does not allow duplicates.

```
set1 = {1,2,3,4,5}
```

Frozenset

A frozenset is a collection of objects that is unchangeable, unordered and does not allow duplicates.

```
frozenset1 = ({1,2,3,4,5})
```

Dict

They are a list of key-value pairs. Also called a "Map" or “Hashtable”. The Values can be accessed using the Keys as indexes.

A dictionary with Str Keys and Int Values.

```
dictionary = {"Jaish": 100, "Subhan":90, "Yasir":80}
print(dictionary["Jaish"])
```

Output: **100**

We can print the dictionary directly using print( ).

```
dictionary = {"Jaish": 100, "Subhan":90, "Yasir":80}
print(dictionary)
```

Output: **{'Jaish': 100, 'Subhan': 90, 'Yasir': 80}**

Dictionary Functions

.get( )

It returns the value of the specified key and if that key doesn’t exist then it returns the second argument.

```
dictionary = {"Jaish": 100, "Subhan":90, "Yasir":80}
print(dictionary.get("JAish", "Does not Exist"))
```

Output: **Does not Exist**

.keys( )

It returns all the keys in the dictionary.

```
dictionary = {"Jaish": 100, "Subhan":90, "Yasir":80}
print(dictionary.keys())
```

Output: **dict_keys(['Jaish', 'Subhan', 'Yasir'])**

.values( )
It returns all the values in the dictionary.

```
dictionary = {"Jaish": 100, "Subhan":90, "Yasir":80}
print(dictionary.values())
```

Output: **dict_values([100, 90, 80])**

.items( )
It returns all the pairs in the dictionary.

```
dictionary = {"Jaish": 100, "Subhan":90, "Yasir":80}
print(dictionary.items())
```

Output: **dict_items([('Jaish', 100), ('Subhan', 90), ('Yasir', 80)])**

Type( ) Function

It is used to get the data type of the variable.

```
x = {1,2,3}
print(type(x))
```

Output: **<class 'set'>**

Type Casting

This is used to convert a variable from one data type to another.

```
x = "123"
y = int(x) #Type Casting
```

This is useful in the input( ) function when we are doing numerical calculations, as it returns a String.

```
x = 10
y = int(input("Enter the number to be multiplied: "))
print(x*y)
```

Output: **Enter the number to be multiplied: 10**
100

Structures

Loops

For Loop

The for loop is used to repeat a block.

```
for i in range(3):
    print("Hello")
```

The range( ) data type is used to specify the Start, Stop and Step indexes.

```
myList = [1,2,3,4,5,6,7,8,9,10,11,12]
for i in range(1,9,2):
    print(myList[i])
```

Output: **2, 4, 6, 8**

Range
It is a data type (in Python 3+) that is an unchangeable sequence of numbers.

While Loop

The while loop is used to repeat a block as long as the condition stays True.

```
x = "0"
while(x=="0"):
    x = input("Enter 0 to continue.")
    print("Hello")
```

Output:
Enter 0 to continue. 0
Hello
Enter 0 to continue. 0
Hello
Enter 0 to continue. 0
Hello
Enter 0 to continue. 1
Hello

The increment/decrement operators can be used to change the value of the condition variable.

```
x = 0
while(x<6):
    print(x)
    x+=1
```
Output: **0, 1, 2, 3, 4, 5**

Break Statement

Used to break out of the loop if a condition is met.
Used to break out of While(True) loops.

```
x = 0
while(True):
    print(x)
    x+=1
    if x>5:
        break
```
Output: **0, 1, 2, 3, 4, 5**

In a nested loop, it breaks the innermost loop.

Continue Statement

Used to skip over the rest of the loop.

```
x = 0
while(x<6):
    if x==3:
        continue
    print(x)
    x+=1
```
Output: **0, 1, 2**

Pass Statement

Used to make a statement; get ignored by the program.

```
x = 0
while(x<6):
    if x==3:
        continue
```

```
    if x==5:
        pass #This does nothing
    print(x)
    x+=1
```
Output: **0, 1, 2**

Nested Loops

Having a loop inside another loop.

Nested For Loops are used to access Multi-Dimensional Lists.

```
list_of_lists_of_lists = [[[1],[2]],[[2,3],[4,5]],[[3,4,5],[6]]]

for i in range(len(list_of_lists_of_lists)):
    for j in range(len(list_of_lists_of_lists[i])):
        for k in range(len(list_of_lists_of_lists[i][j])):
            print(list_of_lists_of_lists[i][j][k])
```
Output: **1, 2, 2, 3, 4, 5, 3, 4, 5, 6**

Selection Structures

If - Elif - Else

It is used to perform certain operations based on conditions.

```
x = 10

if x < 10:
    print('less')
elif x > 10:
    print('greater')
else:
    print('equal')
```

Try - Except - Else - Finally

It is used for Error handling.

Try: Code to be checked.

Except: Code to be run in case of an Error.

Else: Code to be run incase no Error occurs.

Finally: Code to be run regardless if an Error occured or not.

```
try:
    print("Check for Errors")
except:
    print("Runs if there is an Error")
else:
    print("Runs if there are no Errors")
finally:
    print("Runs regardless")
```

Output:
Check for Errors
Runs if there are no Errors
Runs regardless

We can have multiple Except statement for different types of Errors.

```
try:
    print(x)
except NameError:
    print("Define x.")
except TypeError:
    print("Wrong Type")
```

Output: **Define x.**

Errors

1. TypeError
2. NameError
3. ZeroDivisionError
4. IndentationError
5. ImportError
6. RuntimeError
   etc.