Python Notes

AUTHOR: MOHAMMED MOFIZUR RAHMAN

Python Extensions for VS Code:

1. **Python** by Microsoft
2. **Pylance** by Microsoft
3. **Code Runner** by Jun Hun
4. **Path Intellisense** by Christian
5. **Python Indent** by Kevin Rose
6. **Error Lens** by Alexander
7. **Kite Autocomplete: Python, Java Script, etc**. by Kite

Variables

A variable is a container for a value, which can be of various types

'''
This is a
multiline comment
or docstring (used to define a functions purpose)
can be single or double quotes
'''

"""
VARIABLE RULES:
 - Variable names are case sensitive (name and NAME are different variables)
 - Must start with a letter or an underscore
 - Can have numbers but cannot start with one
"""

```python
x = 1
y = 2.5
y = int(y)  # Casting y as an Integer.
z = float(y)
name = 'Mohammed Mofizur Rahman'
is_cool = True
a = x + y
p, q, city = (5, 7.5, 'Dhaka')  # Input multiples

# z will be 2.0 cz y turns to integer so that 2.5 -> 2 then, z = 2.0
print(type(x), y, name, is_cool, a, p, q, city, z)
```

Strings

Strings in python are surrounded by either single or double quotation marks. Let's look at string formatting and some string methods

```
Name = 'Mohammed Mofizur Rahman'
age = 22
# 1 Concatenation
# print('Hello! My name is ' + Name + 'My age is ' + str(age))

# 2 String Formatting
# Formatting: Arguments by position

# print('My name is {myname} and I am {myage} years old.'.format(
#     myname=Name, myage=age))


# 3 Print strind Using F-string
print(f'Hello! My name is {Name} and I am {age} years old.')


# 4 String Methods
a = 'hello World'

print(a.capitalize())

print(a.upper())

print(a.lower())

print(a.swapcase())

# Length of string
print(len(a))

# Replace
print(a.replace('World', 'Everyone'))
```

List

A List is a collection which is ordered and changeable. Allows duplicate members.

```
# 1 Create list
# List allows duplicate members
Numbers = [1, 2, 3, 4, 5]
fruits = ['Mango', 'Apple', 'Grapes', 'Mango', 'Papaya', 'Orange']

# List using Constructor (Like Java Script)
Numbers2 = list((10, 20, 30, 40, 50))
print(Numbers2)

print(Numbers, fruits[2], fruits)

# Length of the list
print(len(Numbers), len(fruits), fruits)
# 2 Append      -> Append adds only 1 member
fruits.append('Musambee')
# 3 Extend      -> Extend adds multiple members as like as al list
fruits.extend(['Jackfruit', 'Watermalon'])
# 4 Remove a member from the list
fruits.remove('Apple')
# 5 Insert an item into a certain position -> Insert Watermelon in index 2
fruits.insert(2, 'Watermalon')
# 6 Remove an item using pop (as like as Stack- push pop)
fruits.pop(2)
# 7 Reverse the list items
fruits.reverse()
# 8 Sort the list Alphabetically
fruits.sort()
# 9 Reverse sort
fruits.sort(reverse=True)
# 10 Change the value
fruits[1] = 'Blueberries'
# Clear a List
fruits.clear()

print(fruits)
```

Tuple

A Tuple is a collection which is ordered and unchangeable. Allows duplicate members.

```
1.
# Create Tuples with duplicate member/item
# Tuples allow duplicate items
fruits = ('Apples', 'Oranges', 'Grapes', 'Oranges')

2.
# Single parameter
fruits2 = ('Apple')
print(type(fruits2))      # Considers as a String

3.
# Using Constructor
fruits2 = tuple(('Apples', 'Oranges', 'Grapes'))
print(fruits, fruits2)
print(type(fruits))     # Considers as a Tuple

4.
# Delete Tuple
del fruits

5.
# Length of tuple
print(len(fruits))

6.
# Check if the item is in the set
print('Apples' in fruits)

7.
# Checking index
print(fruits[0])      # It works


8.
# Tuple can not be clreared   -> Tuples do not have clear attribute
print(fruits)
```

Set

A Set is a collection which is unordered and unindexed. No duplicate members.

```
1.
# Create Set
# A set does NOT allow duplicate items/members
fruits_set = {'Jackfruits', 'Jackfruits',
              'Watermalons', 'Pears', 'Blackberries', 'Pears'}

2.
# Check if the item is in the set
print('Apples' in fruits_set)

3.
# Adding an item
fruits_set.add('Banana')

4.
# Does NOT add an item twice
fruits_set.add('Jackfruits')

5.
# Removing an item
fruits_set.remove('Pears')

6.
# Clearing the set
fruits_set.clear()

7.
# Delete set
del fruits_set

print(fruits_set)
```

Dictionary

A Dictionary is a collection which is unordered, changeable, and indexed. No duplicate members.
Read more about dictionaries at
https://docs.python.org/3/tutorial/datastructures.html#dictionaries

```
1.
# Create dictionary
person = {
    'first_name': 'Mohammed Mofizur',
    'last_name': 'Rahman',
    'age': 22,
    'gender': 'Male'
}

2.
# Create dictionary using Constructor
person2 = dict(first_name='Sara', last_name='Hafiza')
print(person2, type(person2))


3.
# Get value
# 1
print(person['gender'])
# 2
print(person.get('first_name'), person.get('age'))

4.
# Add a member to the dictionary
person['phone'] = '+8801779746565'

5.
# Get Keys
print(person.keys())

6.
# Get Items
print(person.items())

7.
# Copy dictionary
person2 = person.copy()
person2['city'] = 'Chittagong'
print(person2)
```

```
8.
# Removing an item/member
del (person['age'])
person.pop('last_name')
print(person)

9.
# Clear the dictionary
person.clear()  # delete does NOT work for dictionary, so use clear
print(person)


10.
# List of dictionary
person3 = [
    {'name': 'Maria', 'age': 20},
    {'name': 'Maruf', 'age': 22}
]
print(person3, type(person3))

11.
# A person's name print
print(person3[1]['age'], person3[0]['name'])


print(person)
```

List	Tuples	Sets	Dictionary
A collection of items which is Ordered, indexed, and changeable	A collection of items which is Ordered, indexed but **Unchangeable**.	A collection of items which is **Unordered,** and **Unindexed**	A collection of items which is **Unordered**, indexed, and changeable
Allows duplicate members	**Allows** duplicate members	Does **NOT** allow duplicate members	Does **NOT** allow duplicate members
List can be empty **[]**	Tuple can **NOT** be empty	Sets can be Empty **{}**	Dictionary can be Empty **{}**
Have a clear attribute. So, You can clear and delete a List	Does **NOT** have the clear attribute. So, You can **NOT** clear a tuple, but you can delete a tuple	Have a clear attribute. So, You can clear and delete a tuple	

List:
 	
`Numbers = [1, 2, 3, 4, 5]`
 	
`fruits = ['Mango', 'Apple', 'Grapes', 'Mango', 'Papaya', 'Orange']`
	Tuple:
	
	`fruits = ('Apples', 'Oranges', 'Grapes', 'Oranges')`
		Set:
		
		`fruits_set = {'Jackfruits',`
		
		`'Jackfruits',`
		
		`'Watermelons',`
		
		`'Pears',`
		
		`'Blackberries',`
		
		`'Pears'}`
			Dictionary:
			
			`person = {`
			`'first_name': 'Mohammed Mofizur',`
			
			`'last_name': 'Rahman',`
			
			`'age': 22,`
			
			`'gender': 'Male'`
			`}`

Note: *When a set is wanted to clear it gives you an empty set, if you delete a set then the set will not exist.*

Function

A function is a block of code which only runs when it is called. In Python, we do not use parentheses and curly brackets, we use indentation with tabs or spaces

```
1.
# Method: 1
def func(name, age):
    print(f'My name is {name} and I am {age} years old.')


func('Mohammed Mofizur Rahman', 22)    # With Argument
2.
#Method: 2
def func2(name='Mohammed Ilisa Azam Chowdhury', age='52'):
    print(f"My father's name is {name} and he is {age} years old.")

func2()             # Without Argument

3.
# Maths
def func3(num1, num2):
    total = num1 + num2
    print(total)

func3(20, 50)
4.
def func4(num3, num4, num5, num6):
    math = num3+num4*num5-num6
    print(math)


func4(30, 4, 5, 5)


5. Lambda Function

# A lambda function is a small anonymous function.
# A lambda function can take any number of arguments, but can only have one
expression. Very similar to JS arrow functions

# You have to disable prettier extension OR
# You have to edit on "editor.formatOnSave" : false

getsum = lambda num8, num9: num8 + num9
print(getsum(10, 5))
```

Conditionals

If/ Else conditions are used to decide to do something based on something being true or false

```
x = 70
y = 100

1.
# Comparison Operators (==, !=, >, <, >=, <=) - Used to compare values
# Simple If / Else if and / Else
if x > y:
    print(f'{x} is greater than {y}')
elif x == y:
    print(f'{x} is equal to {y}')
else:
    print(f'{y} is greater than {x}')


2.
# Nested If
if x > 50:
    if x < 100:
        print(f'{x} is greater than 50 and less than 100')



3.
# Logical operators (and, or, not) - Used to combine conditional statements
if x > 70 and x < 100:
    print(f'x is equal to {x}')

if x > 70 or x < 100:
    print(f'x = {x}')


if not (x == y):
    print(f'x which is {x}, is not equal to y')
else:
    print(f'x which is {x}, is equal to y')

4.
# Membership Operators (in, not in) - Membership operators are used to test
if a sequence is presented in an object
Numbers = [10, 30, 50, 70, 90]
```

```python
if x in Numbers:
    print(x in Numbers)     # Return type : True or False
    print(f'{x} is in Numbers list')

if x not in Numbers:
    print(x not in Numbers)     # Return type : True or False
    print(f'{x} is not existed in Numbers list')


5.
# Identity Operators (is, is not) - Compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:
if x is 70:
    print(x is 70)     # Return type : True or False
    print(f'{x} is equal to 70')

if x is not 70:
    print(x is not 70)     # Return type : True or False
    print(f'{x} is not equal to 70')
```

Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

```
people = ['Mofiz', 'Mahjabin', 'Farzana', 'Abdullah', 'Tasnim']

1.
# Simple for loop
for person in people:
    # print(f'Current person: {person}')
    if person == 'Abdullah':
        # break      # Stops where Abdullah found
        continue     # Skip Abdullah then continue to the next index
    print(f'People : {person}')

print(len(people))


2.
# Range
family = ['Ilias', 'Beby', 'Mofiz', 'Farzana', 'Abdullah', 'Tasnim']
for person in range(len(family)):
    print(family[person])
    print(f'Current person: {family[person]}')


3.
# Range Numbers
for i in range(0, 10):   # for(i=0; i>10; i++)
    print(i)
    print(f'Numbers: {i}')


4.
# While loops execute a set of statements as long as a condition is true.
count = 0
while count <= 10:
    print(count)
    count += 1
```

Modules

A module is basically a file containing a set of functions to include in your application.

There are:

1. Core python modules,
2. Modules you can install using the pip package manager (including Django),
3. Custom modules.

```
# [1]core python modules

1.
import datetime            # Here datetime is a module/file
today = datetime.date.today()   # Datetime is a module, date.today() is a
method/function
print(today)


2.
from datetime import date
today = date.today()
print(today)


3.
import time
timenow = time.time()   # Here, time is a module and time() is a method/
function
print(timenow)


4.
from time import time
time1 = time()
print(time1)
```

Custom Module

Validator.py:

```
# Example of a custom module to be imported

import re               # re = Regular expression module


def validate_email(email):
    if len(email) > 7:
        # Match method checks the signs of an email by regular expressions
        return bool(re.match("^.+@(\[?)[a-zA-Z0-9-.]+.([a-zA-Z]{2,3}|[0-
9]{1,3})(]?)$", email))


def validate_email2(mailing):
    if len(mailing) > 7:
        return bool(re.match("([a-zA-Z0-9_\-\.]+)@((\[[0-9]{1,3}\.[0-
9]{1,3}\.[0-9]{1,3}\.)|(([a-zA-Z0-9\-]+\.)+))([a-zA-Z]{2,4}|[0-9]{1,3})(\]?)",
mailing))
```

modules.py:

```
# Import Custom Module
from validator import validate_email2
from validator import validate_email


1.
email = 'mohammed@gamil.com'

if validate_email(email):
    print('Email is valid!')
else:
    print('Email is invalid! Please enter a valid email address')


2.
email2 = 'mohammedeiman0177@gmail.com'    # Variable: email2
# Validate_email2(variable pass here)------- if true then print
if validate_email2(email2):
    print('Congratulation! Your email is valid.')

else    # if false then print sorry
    print('Sorry! Your email is invalid. Please, emnter a valid email address.')
```

Classes

A class is like a blueprint for creating objects. An object has properties and methods(functions) associated with it. Almost everything in Python is an object

Class-7 (Class)			
Name (Properties)	***id***	***age***	***Cgpa***
Afia Tasnim **(Objects)**	530722	12	3.5
Tashpia Alam Piyasha	060722	12	3.6
Nafeesa Tabassum Dipu	250722	13	3.45

```
# Create class
class class_7:
    # Constructor
    def __init__(self, name, id, age, cgpa):
        self.name = name
        self.id = id
        self.age = age
        self.cgpa = cgpa

# Initialize objects by Roll
Roll_53 = class_7('Afia Tasnim', '530722', '12', '3.5')
Roll_06 = class_7('Ipshita Alam Piyasha', '060722', '12', '3.6')
Roll_25 = class_7('Nafeesa Tabassum Dipu', '250722', '13', '3.45')

print(Roll_53.name, Roll_53.id, Roll_53.age, Roll_53.cgpa)
print(Roll_06.name, Roll_06.id, Roll_06.age, Roll_06.cgpa)
print(Roll_25.name, Roll_25.id, Roll_25.age, Roll_25.cgpa)

# Print by using f-string
print(f'My name is ' + Roll_53.name + ' and my id is ' +
      Roll_53.id + '. I am ' + Roll_53.age + ' years old.')
```

Create a class for user and override it with Customer.

```
# Create class
class user:
    # Constructor
    def __init__(self, name, id, cgpa):
        self.name = name
        self.id = id
        self.cgpa = cgpa

    # Method/Function to pass the value of self object- person1 or person2
or person3 or etc...

    def greeting(self):
        return f'My name is {self.name} and my id is {self.id}'


# Initialize user object
person1 = user('Mohammed Mofizur Rhaman', '19-40120-1', '3.66')
print(type(person1))
print(person1.name, person1.id, person1.cgpa,)
print(f"I am " + person1.name + ". My id is " + person1.id +
      " and my cgpa is " + person1.cgpa+".")
print(person1.greeting())


# Overridding/Extending user class to customer class
class customer(user):
    # Constructor
    def __init__(self, name, id, cgpa):
        self.name = name
        self.id = id
        self.cgpa = cgpa
        self.balance = 0
        # self.starting_date = '10 Oct, 2022'

    def set_balance(self, balance):
        self.balance = balance


person55 = customer('Eiman', '19-40120-1', '3.66')
person55.set_balance("$5000")   # Update balance

print(person55.name, person55.id, person55.cgpa, person55.balance)

# Method can be called from overridden methods
print(person55.greeting())
```

Files

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

```
# Create a file by Open command/method, name it as Eiman.txt and put the
file on myfile instance/object. Object name can be any.

myfile = open('Eiman.txt', 'w')  # Mode = Write(w)


# Get some information
print('Name: ', myfile.name)
print('Is closed: ', myfile.closed)
print('Opening mode: ', myfile.mode)


# write into the file
myfile.write('I love Python ')
myfile.write('and I am learning Python.')
myfile.write(' I also like JavaScript.')
myfile.close()

# Append/Add into the file
myfile = open('Eiman.txt', 'a')
myfile.write(' This is an append text.')
myfile.close()


# Read from a file
myfile = open('Eiman.txt', 'r+')
readTheText = myfile.read(500)
print(readTheText)
```

JSON

JSON is commonly used with data APIs.

Here how we can parse JSON into a Python dictionary is shown below:

```
import json

# Simple JSON
Mofiz_person_1 = '{"first_name": "Mohammed","last_name": "Eiman","age":
"22","cgpa": "3.66"}'

# Print the properties of Mofiz_person_1 object through loading the jason
dictionary using json.loads
print(json.loads(Mofiz_person_1))

# Or create an object named user and pass the loads of json into the user
object/instance
# This is a system to find the dictionary from the JSON format
Elias = json.loads(Mofiz_person_1)
print(Elias)

# Make JSON format from a dictionary
car_Mofiz = {'made by': 'Toyota', 'Model': 'Corolla', 'Year': '2022'}
print(json.dumps(car_Mofiz))

Corolla2022 = json.dumps(car_Mofiz)
print(Corolla2022)
```