8:42 PM
BY NASIR HUSSAIN

MON 3-AUG-2020

"PYTHON":-

Python is interpreted language.

Print function:-

```
print("Hello Python")
```

VERSION OF PYTHON:-

- Version 2
- Version 3

PRINT:-

```
print(1000) # integer        output → 1000
print(123, 456)              output → 123 456
print("python") # string     output → python.
```

seperator:

- print("pakistan", 'python', "piaic", sep=" ")
- print("pakistan", end=" ") or end=\t tab or \n line

VARIABLES:-

hold a value that may change

```
a = 10
print(a)
type(a)    output → integer.
```

example

```
student_name = "Zohaib"
father_name = "Shabbir"
print(student_name + father_name)
```

example

```
value 1 = 100
value 2 = 10.5

sum = value 2 - value 1
print(sum)
```

example

```
value 1 = 100
value 2 = 100
sum = value 1 / value 2
```

output by default float.

OR

```
sum = value 1 // value 2
```

output on integer

⊙ '+' operator is for concatenation.

IF ELSE STATEMENT :-

```
a = 100
b = 200
if a<b:                    # returns true
    print("a is less than b")
    print("Hello A")
    print("Hello B")
Print ("Hello Python")

if b<a:                    # return false
    print("b is less than a")
print("Pakistan")
```

LIST :-

data structure in python that is a mutable or changable, order sequence of element.

example:

```
friends = ['Hamza', 'Fatima', 'Azhar', 'Laiba', 'Ali',]
```

it is not necessary that data type is of single type. like in Arrays.

example

alist = ["nasir", "ali", faisal"]

index 0, 1, 2

To check length — output

len(alist) — 3

APPEND FUNCTION

Adding a member to the list

fruits.append("apple") : output ['apple']

INSERT FUNCTION:-

fruits.insert(3, "mango")

EXTEND FUNCTION → For multiple values.

fruits.extend(["apple", "Cherry", "bluebery"])

COUNT FUNCTION

fruits.count("orange") output 5

INDEX FUNCTION

fruits.index("Cherry") output 9

CLEAR FUNCTION:-

fruits.clear()

COPY FUNCTION — By reference fruits3 = fruits

fruits.copy() — By value fruits2 = fruits.copy()

: Pass by reference means it will reflect the values from the list if any value is added or deleted.

REMOVE FUNCTION index

```
del fruits2[1]
```

FINDING VALUES IN LIST :-

```
arr = [1, -3, -4.5, 32, 0.2]
arr.index(1)
```

SLICING ELEMENT

```
arr = [1, 4, 2, 78, 45, 23, 89]
print(arr[1:4])        output => (4, 2, 78)
#index      -5   -4   -3   -2   -1
students = ["Ali", "Faisal", "Saleem", "Hamza", "kashf"]
#index      0    1    2    3    4
                 index   output
print(students[0])       Ali
print(students[-5])      Ali
#students[start : end+1]
                         output
students[1:3]            ["Faisal", "Saleem"]
```

students [-4, -2] ['faisal', 'saleem']

STEP WISE

nums = [0 : 16 : 1]

output

[0, 2, ... 16]

nums = [0 : 16 : 2]

output

[1, 3, 5, 7 ... 15]

nums = [: : 5]

output

[1, 6, 11, 16]

REMOVING ELEMENTS :-

del arr[2]
index

arr.remove(2)

POP

It removes the last element of list but it stores

arr.pop() it removes last value

app.pop(2) it remove the value of index 2

TUPLES :-

Tuples are just like Python list except that they are immutable. you can not add, delete and change elements after the creation of Tuple instance.

```
tpl = (1, 4, 5, True)
print(tpl[1])          output → 4
tpl[1] = 10            output → error
```

PART 2

FOR LOOP

```
arr = [1, 2, 3, 6, 67, 23, 45]
for elem in arr:
    if elem = 6
```

BREAK :-

```
        break  # loop will be terminated
               as it encounters 6.
```

CONTINUE :-

```
    if elem % 2 == 0:
        continue  # loop will skip here to next
                  iteration.
    print(elem)
```

FOR LOOP PRACTICE

- for a in range(5): #5 will be exclusive
  print("Zohaib")

- for number in range(1, 10):
  print(number) # output 1, 2, 3, 4, 5, 6, 7, 8, 9

STEPS

- for number in range(1, 10, 2):
  print(number) # output 1, 3, 5, 7, 9

REVERSE ORDER :-

- For number in range(10, 1, -2):
  print(number) # output 10, 8, 6, 4, 2

- for city in cities
  print(f"The city under consideration is {city}")

output

The city under consideration is khi
" " " " multan
" " " " quetta

output

- country = "Pakistan"
  for char in country:
  print(char)

P
a
k
i
s
t
a
n

country = "Pakistan", "American"

```
for char in country
    print(char)
```

output

Pakistan
American.

FOR LOOP'S BREAK & CONTINUE

①
```
for number in range(10):
    if number % 3 == 1:
        break
    print(number)
```

②
```
for number in [5, 7, 11, 90]
    if number % 2 == 0
        break.
    print(number)
```

③
```
for number in range(10):
    if num == 7 or num == 4
        continue
    print(num)
```

output
0
1
2
3
5
6
8
9

NESTED LOOP :-

```
for a in range(5):
    print("Inner loop begin")
    for char in "China":
        print(a, char)
```

output :-

```
Inner loop begin
0 C
0 h
0 i
0 n
0 a
Inner loop begin
1 C
1 h
1 i
1 n
1 a
Inn
```

Table number example :-

```
Table number = int(input("Enter table no"))
for a in range(1, 10):
    print(f"{table number} * {a} = {tablenumber * a}")
```

Another example of Table :

```
tables = Int(Input("Enter table"))
for table in range(2, tables+1):
    for num in range(1, 11):
        print(f"{table} * {num} = {table * num}")
```

TYPE CAST :-

We convert one data type to another and input function return user's data as string so we need to type cast from strings to numbers and vise versa

```
user_input = int(input("Enter something"))
type(user_input)
float(6)    output  6.0
```

CHANGE CASE :-

Change string case like upper, lower, title.

```
str_data = "I AM A HUMAN"
print(str_data.lower())      lower case.
print(str_data.upper())      upper case
print(str_data.title())      title case.

name = "Pakistan"
name.upper()
```

DICTIONARIES :-

It stores key value pairs e.g.

```
student = {'name': 'Zaid', 'Class': 'AI', 'Program':
'PIAIC', 'Age': 42}
```

Note

Pls note that keys must be blw single or double quotation if they are string type, every pair is seperated from other pair by a **comma**, and every pair of key value is seperated by a **colon**.

Example

myDict = {"name": "Zohaib", "age": 30, "gender": "M"}

len(myDict)

ADDING A NEW KEY IN DICT.

myDict = ["email"] = "abed@gmail.com"

it overwrite if we give same key and diff. value.

ACCESSING INFO FROM DICTIONARIES :-

print(student['name']) output → error as name does not exist 'Name' right.

- newDict = {1: "Asad", 2: "Saad"}

newDict[1]

- del newDict[1]

To check value :-

1 in newDic output True.

"age" in myDict output True.

ITERATING OVER INFO FROM DICTIONARY

It provides 3 methods to iterate over dictionary.

- Value — student.value() — print(value)
- keys — student.key() — print(key)
- key value pair. student.items() — print(key, value

- Dictionary can store any thing like:
  - Dictionary can contain list.
  - Tuple
  - Any type primitive or user defined
  - Dictionary, a dictionary can store an other dictionary as its value of combination of these.

LIST OF DICTIONARY:-

We are creating a list in which each member will be dictionary.

```
student=[ ]
students.append({'Name': 'Ali', 'Class': 'AI',
  'Campus': 'PIAIC Campus 1'})
students.append({'Name': 'Jinah', 'Class': 'AI',
  'Campus': 'PIAIC Campus 2'})
```

2:08PM 20-2-2020

RAID

By NASIR HUSSAIN

PYTHON PART 3

THU 13-AUG-2020
8:55 AM

FUNCTIONS :-

Functions are a way to acheive modulaxility and reusability in code.

Modulaxility :-
A process of subdividing a computer program into seperate sub-programs.

Reusability :-
Using of already developed code according to our requirement without writing from the ~~se~~ scratch.

DEFINE FUNCTION (FUNCTION PART 2)

A keyword "def" is used and after that name of function we supply pair of parentheses and a colon sign eg.

```
def add():
    number1 = int(input('Enter a value'))
    number2 = int(input('Enter another value'))
    print(number1 + number2)
```

CALLING FUNCTION (FUNCTION PART 3)

Simply call function e.g. from above exaple

```
add()
```

(FUNCTION 4)

PASSING INFO POSITIONAL ARGUMENTS:-

```
def add(number1, number2):
    print(number1 + number2)
add(3, 5)
```

passing values to number1 & number2 arguments.

KEYWORD ARGUMENT:- (FUNCTION 5)

If we forgot the position then we use keyword so it will exactly same argument

```
def add(number1, number2)
    print(number1 + number2)
add(number2 = 8, number1 = 8).
```

DEFAULT VALUE PARAMETERS:- (FUNCTION 6)

Sometimes some parameters value are optional but still you need a default value in case if someone does not provide the value to avoid any non deterministic behaviour .e.g.

```
def add(number1 = 0, number2 = 0):
    print(number1 + number2)
add(number2 = 5)
```

- All default parameters must be kept at last

```
def fullname(first, last, middle = " "):
    print(first + middle + last)
```

Dealing with An Unknown No. Of Argument :-

If user provides more arguments then

```
def display_nums(first_num, second_num, *opt_nums)
    print(first_num)
    print(second_num)
    print(opt_nums)
```

Passing Information Back From Them:-

Functions not only perform a given task when they are called, But a function may also return some value to user. This returned value can be assigned, reused and be modified

```
def ans(val1, val2):
    ans = val1 + val2
    return ans, "Hello Python"

result = add(2, 4)
```

USING FUNCTION AS VARIABLE :-

Functions can be used as variable

```
def add(a, b):
    return a+b

def sub(b, a):
    return a-b

result = add(2,3) + sub(2,3)
```

output → 6

LOCAL Vs GLOBAL VARIABLES :-

Local variable are defined inside the function They are not accessible outside the function.

Global variable are the variable defined outside the function and can be accessed and modified in and outside the function.

Local Variable :-

```
def behappy():
    name = "mr. A"
    print(f"{name} is very happy")
```

- behappy()
- print(name) → error its local

Global Variable :-

```
anothername = "mr. B"
def sad():
    print(f"{anotherName} is sad")

sad()
print(anothername)
```

FUNCTION WITHIN FUNCTIONS :-

```
def commissionCalc(sales):
    if sales > 100:
        return sales * 100
    elif sales > 50:
        return sales * 50
    elif sales > 20:
        return sales * 20
    else:
        return 0
def salaryCalc(basic, sales):
    grossSalary = basic + commissionCalc(sales)
    print(f"Your gross salary is {grossSalary}")

salaryCalc(50,000, 150).
```

WHILE LOOP:

It allow user to terminate loop by setting flag.

Example

```
a = 0
while a < 10 :
    print ("This is while loop printing")

    a += 1
```

Example #2

```
flag = True
favFoods = [ ]
while flag :
    userinput = input ("Enter fav. food")
    if userinput == "Q" :
        flag false

    else:
        favFoods.append (userinput)

favFoods
```

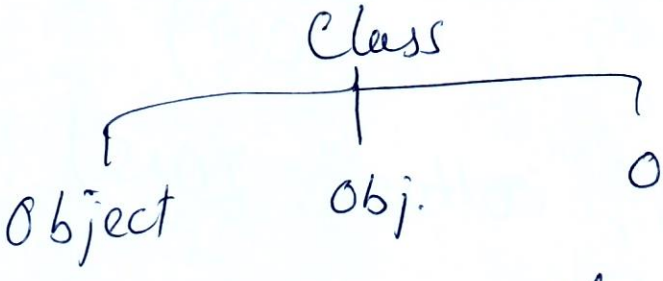
CLASS :-

Python is an object orienled programming language.

Programmers use classes to keep relaled thing together. This is done using keyword "Class" which is a grouping of

- Class is a model
- Class is a blueprint (map) of anything.
- A Templale.
- A Class may also be defined that can be followed by to create object and instance

Objects :-



- A class may hold attributes (variable)
- A class may hold behaviour (functions)

```
Class Car():
    # attributes >> are variable in programming
    def __init__(self, make, model, year):
        self.make = make.
        self.model = model
        self.year = year.
    # behaviour >> are functions in programming.
    def descriptionCar(self):
        print(f"The make of car is {self.make}")
        print(f"The make of Car is {self.model}")
        print(f"The make of Car is {self.year}")
    def move(self):
        print(f"{self.make} is moving with speed")
    def applyingBreak(self):
        print(f"{self.model} has applied the break")
```

Creating Object of Class:

```
Car1 = Car("Honda", "Civic", "2019")
Car2 = Car("Suzuki", "Cultus", 2015)
```

Change attributes

```
Car.make = "Yamaha"
```

Two way to change attributes

1) Direct hit the attributes.
2) Via function (get set)

DATA FILES :-

Modes

1) reading mode.
2) writing mode.
3) append mode.

READ :-

```
with open ("testing Data Files.txt", "r") as file:
    content = file.read
print(content)
```

WRITE :-

```
with open ("testing Data Files", "w") as file:
    file.write("Hello Data File")
```

output → check file testing Data Files

APPEND :-

```
with open ("testing Data Files", "a") as file:
    file.write ("This is new data")
```

r+ mode, w+ mode :-

```
with open ("myNewFile.txt", "w+") as file
    file.write ("This file does not exist")
    print (file.seek(0))
```

- In r+ mode, it will not create file.

MODULES :-

Create a file abc.csv.

```
import csv
with open("abc.csv") as file:
    contents = csv.reader(file)
    for content in contents:
        print(content)
```

JSON FILE :-

Java Script Object Notation

write = dump json.dump
read = load json.load

WRITING DATA TO JSON FILE :-

```
import json
with open("Jsonfile.json", "w") as file:
    json.dump([11, 22, 33, 44, 55], file)
```

READING DATA FROM JSON FILE :-

```
with open("Jsonfile.json", "r") as file:
    json.load(file)
```

Writing data dictionary in Json file:

```
customer_29876 = {"first name": "David",
                  "last name": "Elliott",
                  "address": 4803 Wellesley St"}

with open("jsondic.json","w") as f:
    json.dump(customer 29876, f)
```

For Reading

```
with open("jsondic.json", "r") as file
    con_len = json.load(f)
```

→ con_len

EXCEPTIONS :-

Exception are run time errors.

- Python uses special objects called exceptions to manage errors that arise during a program's execution. whenever an error occurs that make python ensure what to do next, it creates an exception object.
- Exceptions are handled with try-except blocks
- A try except block asks Python to do something, but it tells Python what to do if an exception is raised

When you use try-except blocks, your programs will continue running even if they starts to go wrong.

Example 1:

```
try:
    value1 = int(input("Enter a number"))
    value2 = int(input("Enter another no."))
    ans = value1/value2
except Exception as e:
    print(f"Exception handled >> {e}")
else:
    print(ans)
finally("finally will always run")
```

Example 2:

```
a = [ ]
try:
    5/0
    c = a[3]
except:
    print("Exception caught")
else:
    print("no exception")
finally:
    print("finally will always run")
```

(b)

```
try 5/5
    c = a[3]
except ZeroDivisionError:
```

M. QASIM 9:54 PM Wednesday

MTA 98-381 :- MICROSOFT PYTHON EXAM

```
a = 23
a
→ 23
type(a)
→ int
type(23)
→ int
```

```
a = 24.6
type(a)
→ float
a = "24"
type(a)
→ str
type(True)
→ bool
```

```
type(False)
→ bool
name = ["Ali", "Hamza", 7, True]
type = (name)
→ list
```

square brakets

0 1 2 3 4

-4 -3 -2 -1

TUPLE

```
t1 = (2, "Ali", "Hamza")
type(t1)
type(t1)
→ tuple
```

DICTIONARY

```
d1 = {"Zohaib": "Ahmed", "name": "Shabbir"}
type(d1)
→ dict
```

SET

```
a = {"A", "A", "A", "C", "B"}
{A, B, C}   type(a) → set
```

Data Types Indexing & Slicing

```
a = "23"
print(type(a))
```

Type Cast

```
a = int(a)
print(type(a))

a = float(a)
print(type(a))
```

Remember the data type sequence while converting str → int to float

```
a = "23.4"
print(type(a))
a = float(a)
print(type(a))
a = int(a)
print(type(a))
```

str → float → int.