week 1

Variables
Data types
operators
weekly Test

week 2

Condition statement
Loops
Application of
Strings
Lists
weekly Test

week 3

Application of Tuples, sets, Dictionary
Functions
Modules
weekly Tests

week 4

Try and Exception
Python developer resume preperation
Interview Questions
Final test.

why python?

Simple syntax & Easy to understand
To do task efficiently
Interpreter language.

what is meant by programming?

Teach a computer to do some task

what is language?

To perform that task is called language.

Datatypes:

Datatypes are used to define the type of the variable.

1. Numeric Datatypes
   - Int (1)
   - Float (1.2)
   - complex (1+1j)

2. Sequence Datatypes
   - List [ ]
   - Tuple ( )
   - range - 1 - 10

3. String Datatype - ("Hari")

4. Set Datatype - {1, 2, 3, 4}

5. Mapping Datatype - Dictionary

{a:100} # key : value.

The key value pair is called Dictionary

6. Bool Datatype :-

Checking whether the condition is satisfied by bool datatype.

```
a = 10
if a == 10:
    print ("yes")        => False.
else
    print ("NO")
print (a == 12)
```

Variable :

Variable is used to store some value in memory.

a = 10

Here, a is variable

Dictionary :

Key and value pair is called Dictionary

Index :

Indexing helps us to extract some particular information for the element.

String :

string is used to represent sequence of the character.

```
a = 'hi'
b = "hi"
c = '''hi'''
```

} all are valid string types

List all string functions :

len()
lower()
upper()
Capitalize()
Title()
Strip()
Split()
join()
replace()
startswith()
endswith()

len() - len() helps to return the length of the string

lower() - lower() helps to convert a string into lowercase.

upper() - upper() helps to convert a string into uppercase

capitalize() - helps to change the first element of the character into capital.

title() - helps to change the all first element of the character into capital.

strip() - helps to remove the white/leading spaces & trailing

split() - helps to split into multiple parts based on requirement / split into a list where each word is list item.

join() - helps to join the elements based on requirement.

replace() - helps to replace the character/element

startswith()
endswith() - To search the value which is present in the document or not.

List [ ]

List is a sequence datatype, in which we can change / modify the element. Hence it is mutable.

It is dynamic
List consist of Duplicate values

What are the operations involved in List :

Creating a list
Accessing a Element
Modify the element
Adding the element
removing the element
search and counting
Combining & Extending

List operations :

- apend( )
- pop( ) (By indexing)
- remove( )
- del ( )
- len( )
- count( )
- insert( )
- push( )
- sort( )
- copy( )

append() - to add one element in the end of the function

pop() - to remove one element in the specified position of the function. using indexing

remove() - to remove any element in any place of the function.

count() - to check the length of the particular element in the function.

len() - to check the length of the whole function.

del() - to delete the whole function.

insert() - inserting the element into the function by indexing.

push() - inserting the element in to the function.

sort() - to change the element in an ordered manner (Ascending / Descending)

copy() - copy the function with values.

Tuple ( ) :

Tuple is also a sequence datatype, in which we cannot change / modify the element. Hence it is immutable

It is static.

we can only add / multiply values in tuple.

a = (1, 2, 3, 4)

Eg :-

1. a * 3
2. a + (1, 3, 5, 6)

Tuple operations :

len() - It helps to find the length of the function

min() - It helps to find the minimum value inside the function.

max() - It helps to find the maximum value inside the function.

del() - It helps to delete the function

Indexing - used to specify the place of the element.

what is operators ?

That allows you to perform different types of operation on variables and its values.

Types of operators :

1. Arithmetic operators.

   (+, -, /, %, //, **)

2. Assignment operators

   (=, +=, -=, *=, /=, //=)

3. comparision operators

   (==, <=, >=, !=, <, >)

4. Logical operators

   (and, or, not)

5. Identity operators.

   (is, is not)

6. Membership operators

   (in, not in)

7. Bitwise operators

   (&, |, ^, ~, <<, >>)

Dictionary :

```
emp_info = {
        "stalini" : 22 ,
        "hari"    : 23 ,
        "pavithra" : 23
          }
```

```
emp_info ["hari"]                    ⇒ 23

if emp_info ["hari"] = 23 :          ⇒ yes
      print ("yes")

del emp_info ["hari"]                ⇒
emp_info                             {'stalin' : 22, 'pavithra' : 23}

emp_info.keys()                      ⇒ dict_keys(['stalini', 'pavithra'])

emp_info.value()                     ⇒ dict_values([22, 23])

emp_info.items()

                                     ⇒ dict_items([('stalin', 22),
                                                   ('pavithra', 23)])
```

what are the methods in dictionary?

get()
pop()
del
keys()
values()
items()

Sets :

Sets is an unordered collection of unique elements.

Sets does not allow duplicate values.

a = {1, 2, 3, 4, 5, 6}
a.add(9) ⇒ {1, 2, 3, 4, 5, 6, 9}

a.remove(9)
a.discard(6) ⇒ {1, 2, 3, 4, 5}

remove() - remove the particular value. (last element)

discard() - check the value which is there or not.

Condition statement :

1. If
2. If else
3. If elif else.

Eg: Traffic signal colour

first statement → If

Inbetween statement → elif

last statement → else.

input()

we use the input() to take the input from the user.

```
colour = input ("please enter colour : ")
```

short hand syntax (or) one hand syntax :

* print ("stopvehicle") if colour = "Red" else print ("start vehicle")
* if colour == "RED" : Print ("please stop vehicle")
* pfint ("please stop the vehicle") if colour == "RED"

if ↙

ifelse ↙ print ("ready to go") if Coulour == "yellow"

if elif else ↙ else print ("stop the vehicle")

note: no 'elif' word in one hand syntax

Nested if else :

```
food = "mutton"
var = "keema"
if food = "mutton":
    if var = "keema":
        print("give keema")
    else:
        print("mutton biryani")
else:
    print("chicken biriyani")
```

For Loop :

It is used for iterating actions over the sequence.

Sequence (List, Tuple, range)

Eg:-

1. for i in range(10): => 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
   print(i)

2. for i in range (2, 5) => 2, 3, 4
   print(i)

3. for i in range (5)
    print ("hari \n")

=>
hari
hari
hari
hari
hari

4. List = ["shalini", "hari"] * 2
for i in List
    print (i)

=>
shalini
hari
shalini
hari

note: (remember taking things from bucket - Eg)

short hand syntax - For loop

normal => List = [1, 2, 3, 4]
for num in List:
    print (num * 2)

short hand => List = [1, 2, 3, 4]
doubled = [num * 2 for num in List]
print (doubled)

(output : 2, 4, 6, 8, 10, 12, 14)

while loop :

with the while loop we can execute a set of statements as long as the condition is true.

1.
```
clr = ["red", "blue", "green", "orange", "pink"]
while clr == "pink":
    print(clr)
```

2.
```
i = 5
while i>5 :
    print(f"{i} hari")
    i+1 = 1   # i = i+1
```

3.
```
a = 0
while a< 1:
    print(a)
    a= a+1
    print(a)
```
⇒ 0
1

4.
```
i = 1
while i<8 :
    print(i)
    i =i+1
```
⇒ 1
2
3
4
5
6
7

5.
```
i = 1
while i<8 :
    print(i)
    i = i+1
else :
    print (f"i value is not less than 8 : {i}")
```

=>
```
1
2
3
4
5
6
7
i value is not less than 8 : 8
```

pass :

It helps to do nothing

break :

with the break statement we can stop the loop even if the while condition is true.

continue :

with the continue statement we can stop the current iteration, and continue with the next iteration.

pass :

```
a = 0
while a == 0:
    print(a)
else:
    pass
```

```
i = 0
while i < 100:
    if i == 50:
        print ("take a break")
```

continue :

```
i = 0
while i < 100:
    i = i + 10
    print (i)
    if i == 50:
        print ("take a 10 min break")
        continue.
```

⇒

```
10
20
30
40
50
take a 10 min break
60
70
80
90
100
```

```
i = 0
while i < 100:
    i = +10
    if i == 50:
        print ("back to home")
        break
    else:
        print ("continue the travel")
```

break

```
i = 0
while i < 100 :
    i = i+10
    print(i)
    if i == 50 :
        print ("back to home")
        break.
```

=>
```
10
20
30
40
50
back to home.
```

while loop + if condition:

```
i = 1
while i < 10 :
    print(i)
    if i == 5 :
        print ("give 2 tickets")
    i = i+1
```

=>
```
1
2
3
4
5
give 2 tickets
6
7
8
9
```

while loop + else condition:

```
i = 2
while a < 7 :
    print ("a is less than 7")
    a = a+1
else:
    print ("a value is not 7")
```

=>
```
a is less than 7
a is less than 7
a is less than 7
a is less than 7
a is less than 7
a is less than 7
a value is not 7
```

while loop + for loop :

Functions :

Function is a block of code which returns the specific task. (or) reusing of code.
A block of code, which will run whenever you call.

step 1 → Defining a function

step 2 → calling a function.

Syntax :

```
def greet():          ----> defining
    print ("Hello")
    print ("good morning")
greet()               ----> calling
```

①
```
def add(x,y):
    c= x+y
    print(c)
add (5,4)
```
⇒ 9

②
```
def add (x,y)
    c=x+y
    return c
result = add (5,4)
print (result)
```

③ def add_sub (x, y):

```
    c = x+y
    d = x-y
    return c, d
result1, result2 = add_sub (5,4)
print (result1, result2)
```

- - - - - -

④ def sum_num (a, b, c, d):

```
    sum_num = a+b+c+d          => 10
    return sum_num
sum_num = sum_num (1,2,3,4)
print (sum_num)
```

⑤ def movie_fct (movie, number):

```
    total_amt = movie * number
    return total_amt
jailer = movie_fct (250, 5)
jailer
```

Syntax:

```
def function (arg1, arg2):
    fast = arg1 * arg2
return fast.
```

Difference between print() and return():

print() - It will display the result but does not make it usable elsewhere

return() - It will be return the result to be displayed/used in the calling code.

```
def add(1,2):
    result = 1+2
    print(result)
```

=> 3

The value `3` is not stored in `result`, when it is called.

```
def add(1,2):
    result = 1+2
    return(result)
```

=) 3

The value `3` is stored in `result`, when it is called.

The above is the difference between both.

Types of Function :

* 1. Built in function --- len(), max, sort() ...
* 2. User defined function
  ---- function returned by user
  3. lambda function --- lambda x : 4
* 4. recursive function --- call themself
* 5. higher order function --- map()
  6. Decorator function
  7. classic and static method
  8. library function --- math (sqrt, sum ...)

Argument :

Argument contains some information, which helps to run the function.

function (arg1, arg2)

Return :

It is a kind of output of the function

Eg:-

```
def cravy(meat):
print(f"this cravy is called {meat} cravy")
```

↓ Format

↓ Dynamic (changeable)

```
cravy(meat = "chicken")
```

=> this cravy is called chicken cravy

when multiple arguments need to be used :

Normal coding →

```
main "rice"
cravy "pappu"
print(f"my morning breakfast is
        {main} with {cravy}")
```

=> my morning breakfast is rice with pappu

using function →

```
def breakfast(main, cravy):
food = f"my morning breakfast is
        {main} with {cravy}"
return food

breakfast("rice", "pappu")
```

=> my morning breakfast is rice with pappu

*arg

①
```
def food(*foods):
    print("my dinner is", foods[1])

food("rice", "chappathi", "mutton cravy")
```

=> my dinner is chappathi

②
```
def food(*foods):
    print("my dinner is" + foods[0])
    print("my dinner is" + foods[1])
    print("my dinner is" + foods[2])
    print("my dinner is" + foods[3])

food("rice", "chappathi", "mutton cravy",
                          "chicken")
```

=> my dinner is rice
my dinner is chappathi
my dinner is chicken
my dinner is mutton cravy

Functions using for loop :

```
def food(*foods):
    word = "my dinner is"
    for i in foods:
        print(word + i)

food("rice", "chappathi", "mutton", "chicken")
```

=> my dinner is rice
my dinner is chappathi
my dinner is mutton
my dinner is chicken.

Functions using Dictionary :

```
def random_value(n):
key = n
key1 = n*n
key2 = n+n
return key, key1, key2

random_value(6)
```

=> (6, 36, 11)

** kwargs → arbitery keyword argument

Functions using List :

① def stu_name (**lst):

print("student name start with" + lst ["first name"])

stu-name (first_name = "shalini", middle_name = "",
last_name = "ganesan")

=) student name start with shalini

② def fruit_basket (fruits):

print(len(fruits)) *

for i in fruits:

print(i)

fruit_basket (["banana", "apple", "orange", grapes"])

=)

banana
apple
orange
grapes

class :

class is nothing but which intakes multiple functions. Eg:- class is a blue print from which objects are created.

class

```
class myclass :          ------ creating a class
    print ("stalini")
    p = 2
```

object → `obj = myclass()` ------ creating an object

`obj.p` ------ calling an object

Eg:

```
class travel :
    vehicle = " "

road = travel()                    => road
road.vehicle = "road"
print(road.vehicle).
```

```
Bus = travel()
Bus.vehicle = "Bus")
print(Bus.vehicle)
```

object :

Object is a instance of class. calling a class is object

__init__()

To initialize function/class is known as init()

```
class sbu():

    def __init__(self, name, age):

        self.name = name

        self.age = age
```

methods :

Methods are nothing but a functions.

Each object contains method.

```
class food :
    def __init__(self, i1, i2, i3, i4, i5):
        self.i1 = i1
        self.i2 = i2
        self.i3 = i3
        self.i4 = i4
        self.i5 = i5

    def prepare(self):
        if self.i1 = "rice":
            print("stove on")

        elif self.i5 = "masala":
            print("masala coming on")

menu = food("oil", "haldha", "musts", "rice", "masala")
menu.prepare()
```

=) masala coming on

∴ The above code is a class enclosed with a function and object.

Two types of variable:

Class variable - - - - a = "abc"

instance variable - - - - - - init - - (self, date, time)

File Handling:

File Handling is a performing operation which is used to read, write and manipulate information on some files.

operations in File Handling:

Create - x - To creating .txt file.

write - W - To writing some information.

read - R - To read the information which has been written

apend - A - To update some information

open()

open() - to open some files.

create()

```
stu = open("./my_doc.txt", "x")
```

write()

```
stu = open("./my_doc.txt", "w")
stu.write("This is python time")
```

read()

```
stu = open("./my_doc.txt", "r")
print(stu.read())
```

=) This is python time

```
stu = open("./my_doc.txt", "r")
print(stu.read(4))
```

=) this

readline

```
stu = open("./my doc.tx", "r")
print(stu.readline())
```

---- First line will be printed

=) This is python time

apend()

```
stu = open("./my-doc.txt", "a")
stu.write("added few lines")
stu.close()
```

```
stu = open("./my-doc.txt", "r")
print(stu.read())
```

=) This is python time added few lines.

with open :

```
with open("./my-doc.txt", "w") as stu
    stu.write("Hello")
    stu.close()
```

```
with open("./my-doc.txt", "r") as stu
    content = stu.read()
    print(content)
```

=) Hello

close()

close() - to close the opened files.