Python

Python Basics

```
# declaration of a variable
# named num with initial value of 100
# int num = 100; <= c's way to declare a variable
num = 100

# retrieve the value stored in variable num
# print on the console
print(num)

# get the value of num and add it in a string
# f => formatted string (string having a value of a variable)
print(f"value of num = {num}")
print("value of num = {}".format(num))

print(f"value of num = \n {num}")
```

data types

- in python call data types are inferred
- all the data type are automatically assigned by inspecting
- current value present inside a variable

int

```
num = 100
print(f"num = {num}, data type = {type(num)}")
```

float

```
salary = 15.60
print(f"salary = {salary}, data type = {type(salary)}")
```

single line str

```
first_name = "amit"
print(f"first_name = {first_name}, data type = {type(first_name)}")
```

single line str

```
middle_name = 'd'
print(f"middle_name = {middle_name}, data type = {type(middle_name)}")
```

single line str

```
last_name = 'kulkarni'
print(f"last_name = {last_name}, data type = {type(last_name)}")
print(f"full name = {first_name + ' ' + last_name}")
```

multi line str

```
address = """xyz home,
abc area,
pqr city,
12345"""
print(f"address = {address}, data type = {type(address)}")
```

double quote inside double quote

- solution 1: escape the inner double quote(s)

```
dialog = "Arnold once said in movie Terminator 2 that \"Dont worry, I will be
back\""
print(f"dialog = {dialog}")
```

- solution 2: declare the string with single quote

```
dialog = 'Arnold once said in movie Terminator 2 that "Dont worry, I will be
back"'
print(f"dialog = {dialog}")
```

single quote inside single quote

- solution 1: escape inner single quote(s)

```
dialog = 'Arnold once said in movie Terminator 2 that \'Dont worry, I will be
back\''
print(f"dialog = {dialog}")
```

- solution 2: declare the string using double quotes

```
dialog = "Arnold once said in movie Terminator 2 that 'Dont worry, I will be
back'"
print(f"dialog = {dialog}")
```

bool: True and False are default values

```
can_vote = True
print(f"can_vote = {can_vote}, type = {type(can_vote)}")
```

operators

- used for performing operations
- categories
- mathematical operators
- relational operators
- logical operators

mathematical operators

```
n1 = 10
n2 = 20
```

addition

```
print(f"{n1} + {n2} = {n1 + n2}")
```

string addition

```
str1 = "10"
str2 = "20"
print(f"{str1} + {str2} = {str1 + str2}")
```

subtraction

print(f"{n1} - {n2} = {n1 - n2}")

division

- true division: the result will be always in float
- always returns float value

```
print(f"{n1} / {n2} = {n1 / n2}")
```

- floor division: floor value => discarding float decimal part from float value
- always returns an int value

```
print(f"{n1} // {n2} = {n1 // n2}")
```

multiplication

```
print(f"{n1} * {n2} = {n1 * n2}")
```

- repeat of
- string will be repeated 5 times

```
print("-*-" * 60)
```

get the power of operator

```
print(f"square of 6 = {6 ** 2}")
print(f"cubes of 6 = {6 ** 3}")
print(f"square root of 6 = {6 ** 0.5}")
```

```
## modulus operator (%)
print(f"10 % 6 = {10 % 6}")
```

type hinting

integers

```
num1: int = 100
num2: int = 200
```

strings

```
num1 = "100"
num2 = "200"
print(f"num1 + num2 = {num1 + num2}")
```

type conversion 1

implicit type conversion

- smaller data type gets easily converted to larger data type

int -> float

```
n1 = 100
n2 = 15.60
```

integer gets converted to float

```
result = n1 + n2
print(f"result = {result}")
```

explicit type conversion

```
n1 = "100"
n2 = 200
n3 = 15.60
n4 = "14.79"
```

- not possible as int and str can not be added together

```
result = n1 + n2
print(f"result = {result}")
```

string to int conversion

```
int_n1 = int(n1)
print(f"type of n1 = {type(n1)}, type of int_n1 = {type(int_n1)}")
```

```
result = int_n1 + n2
print(f"result = {result}")
```

string to float conversion

```
float_n1 = float(n1)
print(f"type of n1 = {type(n1)}, type of float_n1 = {type(float_n1)}")
result = float_n1 + n2
print(f"result = {result}")
```

int to string conversion

```
str_n2 = str(n2)
print(f"type of n2 = {type(n2)}, type of str_n2 = {type(str_n2)}")
```

Key Points -

- int to float -> float()
- int to bool -> bool()
- int to string -> str()

```
float to int
float to string
float to bool

bool to string
bool to int
bool to float
```

comparison operators

```
n1 = 10
n2 = 9

print(f"{n1} > {n2} = {n1 > n2}")
print(f"{n1} >= {n2} = {n1 >= n2}")
print(f"{n1} < {n2} = {n1 < n2}")
print(f"{n1} <= {n2} = {n1 <= n2}")
print(f"{n1} == {n2} = {n1 == n2}")
print(f"{n1} != {n2} = {n1 != n2}")
```

logical operators

and operation

```
print(f"{True} and {True} = {True and True}")
print(f"{True} and {False} = {True and False}")
print(f"{False} and {True} = {False and True}")
print(f"{False} and {False} = {False and False}")
```

or operation

```
print(f"{True} or {True} = {True or True}")
print(f"{True} or {False} = {True or False}")
print(f"{False} or {True} = {False or True}")
print(f"{False} or {False} = {False or False}")
```

not operation

```
print(f"not of True = {not True}")
print(f"not of False = {not False}")
```

if condition

```
age1 = 10
if age1 >= 18:
    # print("first line in if block")
    print("person1 is eligible for voting")
else:
    print("person1 is not eligible for voting")

age2 = 30
if age2 >= 18:
    # print("first line in if block")
    print("person2 is eligible for voting")
else:
    print("person2 is not eligible for voting")


age3 = 70
if age3 >= 18:
    # print("first line in if block")
    print("person3 is eligible for voting")
else:
    print("person3 is not eligible for voting")
```

function part 1

- function definition
- parameterless function

```
def function1():
    """this is a dummy function. there are no arguments nor it returns any
value"""
    print("inside function1")
```

function call

```
function1()
```

get the docstring from the function

```
print(function1.__doc__)
- print(print.__doc__)
```

```
def function2(param):
    print("inside function2")
    print(f"param = {param}, type = {type(param)}")

function2(10)
function2("test")
function2(True)
function2(10.40)
```

function part 2

naming conventions

- function name should start with lower case letter - special characters like spaces are not allowed - instead of using spaces use underscore (_)

parameterless

```
def function1():
    print("inside function1")
```

calling a function

```
value = function1()
print(value)
```

parameterized function

- type hinting
- non-returning function
- returns nothing

```
def function2(p1: int, p2: int):
    """
    this is a parameterized function which gets the parameters added together

    :param p1: param 1
    :param p2: param 2
    :return: nothing
    """
    print("inside function2")
    print(f"p1 = {p1}, type = {type(p1)}")
    print(f"p2 = {p2}, type = {type(p2)}")
    addition = p1 + p2
    print(f"addition = {addition}")

function2(10, 20)
```

IDE will show a warning for this function call

```
function2("10", "20")

def function3(p1: int, p2: int):
    """
    returns addition of p1 and p2

    :param p1: param 1
    :param p2: param 2
    :return: addition of p1 and p2
    """
    return p1 + p2
```

call the function and capture its return value in addition variable

```
addition = function3(10, 20.50)
print(f"addition = {addition}")
```

write a function to return cube of a number

```
def cube(number: int):
    return number ** 3

c1 = cube(3)
c1 = cube(number=3)
```

function part 3

```
def function1(p1: int, p2: int):
    print(f"p1 = {p1}, p2 = {p2}")
    addition = p1 + p2
    print(f"addition = {addition}")
```

function call

- passing parameters using the parameter positions - the parameters will be passed from left to right - p1: 10, p2: 20

```
function1(10, 20)
```

p1: 20, p2: 10

```
function1(20, 10)
```

named parameters

p1: 10, p2: 20

```
function1(p1=10, p2=20)
```

Functions Practice

```
def function2(name: str, age: int, address: str, email: str):
    print(f"name: {name}")
    print(f"age: {age}")
    print(f"address: {address}")
    print(f"email: {email}")


function2("person1", 20, "pune", "person1@test.com")
function2("pune", "person1", 20, "person2@test.com")

function2(name="person1", age=20, address="pune", email="person1@test.com")
function2(age=20, address="pune", name="person1", email="person1@test.com")
function2(name="person1", address="pune", email="person1@test.com", age=20)
```

Note -

function parameter default value

- parameters having default value are also known as optional type

rule

- you can have the default value(s) from right to left
- the default values of address and email are empty strings

```
def function1(name: str, age: int, address: str = "", email: str = ""):
    print(f"name: {name}")
    print(f"address: {address}")
    print(f"email: {email}")
    print(f"age: {age}")
```

- since the email and address are having default value
- you can skip passing the values for them **'p': p1, address: '', email: ''**

```
function1("p1")
```

'p': p2, address: 'pune', email: ''

```
function1("p2", address='pune')
function1("p3", "", "p3@test.com")
function1("p3", email="p3@test.com")
```

global variable

- variable declared outside of any function

```
# global variable
num = 100

print(f"outside of any function [before function1()], num = {num}")
```

```
def function1():
    print("inside function1")

    # local scope
    my_var = "test value"

    # use num from the global scope
    # global num

    # modifying the global variable
    num = 200

    print(f"num = {num}")

function1()
print(f"outside of any function [after function1()], num = {num}")
```

local/nested/inner function

```
def outer_function(call_inner_function: bool = True):
    print("inside outer_function")

    # function declared within another function
    def inner_function():
        print("inside inner_function")

    if call_inner_function:
        # inner_function can be called only within the same function
        inner_function()

outer_function(call_inner_function=False)
```

- nested function can not be accessed outside of outer function

```
inner_function()
```

lambda function

- also called as anonymous function
-
- empty lambda

```
useless_lambda = lambda: "useless"
```

lambda function eg

```
square = lambda num: num ** 2
print(f"square = {square}, type = {type(square)}")
print(f"square of 2 = {square(2)}")
```

add

```
add = lambda p1, p2: p1 + p2
print(f"add = {add}, type = {type(add)}")
print(f"10 + 20 = {add(10, 20)}")
```

subtraction

```
subtract = lambda p1, p2: p1 - p2
print(f"10 - 20 = {subtract(10, 20)}")
```

multiplication

```
multiply = lambda p1, p2: p1 * p2
print(f"10 * 20 = {multiply(10, 20)}")
```

calculate cube

```
cube = lambda num: num ** 3
print(f"cube of 2 = {cube(2)}")
```

check if number is even or odd

- solution 1

```
def is_odd_or_even(num: int):
    if num % 2 == 0:
        return "even"
        else:
            return "odd"
```

- solution 2

```
return "even" if num % 2 == 0 else "odd"
```

- lambda solution

```
is_odd_or_even = lambda num: "even" if num % 2 == 0 else "odd"
print(f"21 is {is_odd_or_even(21)}")
```

collection: List

- collection of similar or dissimilar values - use [] to create a list

```
def function1():
    # list of numbers
    numbers = [10, 20, 30, 40, 50]
    print(f"numbers = {numbers}, type = {type(numbers)}")

    # list of strings
    countries = ["india", "usa", "uk", "japan"]
    print(f"countries = {countries}, type = {type(countries)}")

    # list of different types
    mixed_type_list = [10, "india", True, 19.48, "usa", "30", 40]
    print(f"mixed_type_list = {mixed_type_list}, type = {type(mixed_type_list)}")

function1()
```

```
def function2():
    # list of numbers
    numbers = [10, 20, 30, 40, 50]

    # get the number of items in the collection
    print(f"size of numbers = {len(numbers)}")

    print(numbers)
```

```
    # append a new value
    numbers.append(60)
    print(f"size of numbers = {len(numbers)}")
    print(numbers)

    # append new values
    # this will add all values together on last index: do not use this method for
this purpose
    # numbers.append([70, 80, 90, 100])
    numbers.extend([70, 80, 90, 100])
    print(f"size of numbers = {len(numbers)}")
    print(numbers)

function2()
```

```
def function3():
    # list of numbers
    numbers = [10, 20, 30, 40, 50]
    print(numbers)

    # add 15 in between 10 and 20
    # add 15 on 1st position
    numbers.insert(1, 15)
    print(numbers)

    # add 25 between 20 and 30
    numbers.insert(3, 25)
    print(numbers)

    # add 35 between 30 and 40
    numbers.insert(5, 35)
    print(numbers)

    # add 45 between 40 and 50
    numbers.insert(7, 45)
    print(numbers)

function3()
```

List Part 1

removing the values

```
def function1():
    # list of numbers
    numbers = [10, 20, 30, 40, 50]
    print(numbers)
```

```
    # remove the last value
    numbers.pop()
    print(numbers)

    # remove the last value
    numbers.pop()
    print(numbers)

function1()
```

```
def function2():
    # list of numbers
    numbers = [10, 20, 30, 40, 50]
    print(numbers)

    # remove value at position 3
    numbers.pop(2)
    print(numbers)

function2()
```

```
def function3():
    # list of numbers
    numbers = [10, 20, 30, 40, 50]
    print(numbers)

    # remove the value 30
    numbers.remove(30)
    print(numbers)

    # list of countries
    countries = ["india", "usa", "uk", "china", "japan"]
    print(countries)

    # remove china
    countries.remove("china")
    print(countries)

function3()
```

List Part 2

```
def function1():
    # list of numbers
```

```
    numbers = [10, 20, 30, 40, 50]
    print(numbers)

    # remove all elements (make the list empty)
    numbers.clear()
    print(numbers)

function1()
```

```
def function2():
    # list of numbers
    # list allows duplicate element
    numbers = [10, 20, 10, 40, 30, 20, 50, 60, 10, 80, 20, 40]
    print(numbers)
    print(f"10 is present {numbers.count(10)} times")
    print(f"20 is present {numbers.count(20)} times")
    print(f"30 is present {numbers.count(30)} times")

function2()
```

```
def function3():
    # list of numbers
    # list allows duplicate element
    numbers = [10, 20, 10, 40, 30, 20, 50, 60, 10, 80, 20, 40]

    # find the first ocurance of 10
    # print(f"10 is present at {numbers.index(10)} position")

    # first param: item you want to find the out the position of
    # second param: index from which you want to start finding the position of
item
    # this line starts searching for 10 from 0th position
    print(f"10 is present at {numbers.index(10, 0)} position")

    # this line starts searching for 10 from 1st position
    print(f"10 is present at {numbers.index(10, 1)} position")

    # this line starts searching for 10 from 3rd position
    print(f"10 is present at {numbers.index(10, 3)} position")

    # try finding all the positions of 20
function3()
```

```
def function4():
    # list of numbers
    numbers = [10, 20, 10, 40, 30, 20, 50, 60, 10, 80, 20, 40]
```

```
    # for..in loop
    # number here will get a value at every index
    # the for loop will run len(numbers) times
    for number in numbers:
        print(f"number = {number}")

function4()
```

```
def function5():
    # list of positions
    # positions = [0, 1, 2, 3, 4]

    # generate list of sequential numbers
    # param 1: start [0] -> from where you want to start ?
    # param 2: stop  [-] -> where do you want to stop ?
    # param 3: step  [1] -> how do you want to get the next value?
    positions = list(range(0, 5, 1))
    print(positions)

    # same as list(range(0, 5, 1))
    positions = list(range(5))
    print(positions)

    # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    # positions = list(range(0, 11, 1))
    positions = list(range(11))
    print(positions)

    # [0, 2, 4, 6, 8, 10]
    positions = list(range(0, 11, 2))
    print(positions)

    # [1, 3, 5, 7, 10]
    positions = list(range(1, 11, 2))
    print(positions)

    # for index in positions:
    #     print("hello world!!!")

function5()
```

```
def function6():
    # list of numbers
    numbers = [10, 20, 10, 40, 30, 20, 50, 60, 10, 80, 20, 40]
    repeat_count_10 = numbers.count(10)
    print(f"repeat_count_10 = {repeat_count_10}")

    # start from zero and find out index of 10, 3 times
    search_position = 0
```

```
    for index in range(repeat_count_10):
        search_position = numbers.index(10, search_position)
        print(f"10 is present at {search_position}")

        print(f"index = {index}")
        # find out value 10 from (search_position + 1)th index
        search_position += 1

function6()
```

```
def function7():
    # list of numbers
    numbers = [10, 20, 10, 40, 30, 20, 50, 60, 10, 80, 20, 40]

    # used to collection positions for value 20
    value = 20

    # list to collect all the positions
    positions = []

    # count of occurance of value
    count = numbers.count(value)

    # initial position to start searching the value
    search_position = 0

    # start getting the value's positions
    for index in range(count):

        # fid out where value is preset in the numbers collection
        search_position = numbers.index(value, search_position)

        # append the searched position into the positions collection
        positions.append(search_position)

        # next time onwards start searching from (search_position + 1)
        search_position += 1

    print(f"{value} is present at {positions}")


function7()
```

```
def function1():
    # list of numbers
    numbers = [10, 20, 10, 40, 30, 20, 50, 60, 10, 80, 20, 40]
    print(numbers)

    # sort the values in ascending order
```

```
    numbers.sort()
    print(numbers)

    # sort the values in descending order in the same list
    numbers.sort(reverse=True)
    print(numbers)

function1()
```

```
def function2():
    # list of numbers
    numbers = [10, 20, 10, 40, 30, 20, 50, 60, 10, 80, 20, 40]
    print(numbers)

    # reverse the values from the list
    # the original list will get updated
    numbers.reverse()
    print(numbers)

function2()
```

```
def function3():
    # list of numbers
    numbers = [10, 20, 10, 40, 30, 20, 50, 60, 10, 80, 20, 40]
    print(numbers)

    # copy all the elements from copy() to the original collection
    # this is also called as deep copy
    numbers_reversed = numbers.copy()

    # this statement will never capy the values from first to second collection
    # this is also known as shallow copy
    # numbers_reversed = numbers

    # reverse the collection
    numbers_reversed.reverse()

    print(numbers)
    print(numbers_reversed)

function3()
```

list indexing

```

```

```
def function1():
    # list of numbers
    numbers = [10, 20, 30, 40, 50]

    # positive indexing
    print(f"value present at 2: = {numbers[2]}")
    print(f"value present at 0th: = {numbers[0]}")
    print(f"last value present at 0th: = {numbers[len(numbers) - 1]}")

function1()
```

```
def function2():
    # list of numbers
    numbers = [10, 20, 30, 40, 50]

    # find out the values using negative indexing
    print(f"value present at -3 = {numbers[-3]}")

    # notes
    # - the last value is present on -1th location
    print(f"last value = {numbers[-1]}")

    # - the first value is present on -len(list)the position
    print(f"first value = {numbers[-len(numbers)]}")

function2()
```

List - slicing

- positions are known - the positions are sequential

```
def function1():
    # list of numbers
    #         [0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9  ]
    numbers = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

    # [30, 40, 50, 60, 70]
    # positions = [2, 3, 4, 5, 6]
    positions = range(2, 7)

    # collect the values from required positions
    numbers_slice = []
    for index in positions:
        numbers_slice.append(numbers[index])

    print(numbers)
    print(numbers_slice)
```

```
function1()
```

```
def function2():
    # list of numbers
    numbers = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

    # [30, 40, 50, 60, 70]
    # [2, 3, 4, 5, 6]
    print(f"numbers[2:7] = {numbers[2:7]}")

    # [70, 80, 90, 100]
    print(f"numbers[6:10] = {numbers[6:10]}")

    # if the stop value is missing then the last position will be considered by
default
    print(f"numbers[6:]   = {numbers[6:]}")

    # [10, 20, 30, 40, 50]
    print(f"numbers[0:5] = {numbers[0:5]}")

    # if  the start value is missing then the first value will be considered by
default
    print(f"numbers[:5]  = {numbers[:5]}")

    # [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
    print(f"numbers[0:10] = {numbers[0:10]}")
    print(f"numbers[0:]   = {numbers[0:]}")
    print(f"numbers[:10]  = {numbers[:10]}")
    print(f"numbers[:]    = {numbers[:]}")

function2()
```

```
def function3():
    # list of numbers
    #         [0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9  ]
    numbers = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

    # [20, 30, 40, 50, 60]
    print(f"numbers[1:6]   = {numbers[1:6]}")
    print(f"numbers[1:6:1] = {numbers[1:6:1]}")

    # get values from even positions
    print(f"numbers[0:10:2] = {numbers[0:10:2]}")
    print(f"numbers[0::2]   = {numbers[0::2]}")
    print(f"numbers[:10:2]  = {numbers[:10:2]}")
    print(f"numbers[::2]    = {numbers[::2]}")

    # get values from odd positions
```

```
    print(f"numbers[1:10:2] = {numbers[1:10:2]}")
    print(f"numbers[1::2]   = {numbers[1::2]}")

    # get the values reversed
    print(f"numbers[::-1]   = {numbers[::-1]}")

function3()
```

```
def function4():
    # get a string reversed
    # "python" => "nohtyp"
    word = "python"

    # string: collection of characters
    print(f"word[0] = {word[0]}")

    # get the word reversed
    print(f"word[::-1] = {word[::-1]}")

    # convert a string into a list of characters
    word_list = list(word)
    print(f"word_list = {word_list}")


function4()
```

tuple part 1

```
def function1():
    # list of numbers
    numbers_list = [10, 20, 30, 40, 50]
    print(f"numbers_list = {numbers_list}, type = {type(numbers_list)}")

    # tuple of numbers
    numbers_tuple_1 = (10, 20, 30, 40, 50)
    print(f"numbers_tuple_1 = {numbers_tuple_1}, type = {type(numbers_tuple_1)}")

    # tuple of numbers
    numbers_tuple_2 = 10, 20, 30, 40, 50
    print(f"numbers_tuple_2 = {numbers_tuple_2}, type = {type(numbers_tuple_2)}")

function1()


def function2():
    numbers_tuple = (10, 20, 30, 40, 50)
    print(f"numbers_tuple = {numbers_tuple}, type = {type(numbers_tuple)}")
```

```
    # add new value         : not applicable
    # remove existing value : not applicable

    # numbers_tuple.count()
    # numbers_tuple.index()

function2()
```

```
def function3():
    # list with one element
    numbers_list = [10]
    print(f"numbers_list = {numbers_list}, type = {type(numbers_list)}")

    # int variable
    # num = 10
    # numbers_tuple = 10
    numbers_tuple = (10)
    print(f"numbers_tuple = {numbers_tuple}, type = {type(numbers_tuple)}")

    # string variable
    # words_tuple = "test"
    words_tuple = ("test")
    print(f"words_tuple = {words_tuple}, type = {type(words_tuple)}")

    # tuple with one element
    numbers_tuple = (10, )
    print(f"numbers_tuple = {numbers_tuple}, type = {type(numbers_tuple)}")
    print(f"size of number_tuple = {len(numbers_tuple)}")

    words_tuple = ("test", )
    print(f"words_tuple = {words_tuple}, type = {type(words_tuple)}")

function3()
```

tuple part 2

```
def function1():
    # person1 = ("person1", "pune", 40, "person1@test.com")
    #           0,         1,     2,   3
    person1 = "person1", "pune", 40, "person1@test.com"
    print(f"name = {person1[0]}")
    print(f"address = {person1[1]}")
    print(f"age = {person1[2]}")
    print(f"email = {person1[3]}")

    print("-" * 80)

    # extract the values from tuple
```

```
    # here name, address, age and email will be created as variables
    # name will get the value at 0th position [name = "person1"]
    # address will get the value at 1st position [address = "pune"]
    # (name, address, age, email) = ("person1", "pune", 40, "person1@test.com")
    name, address, age, email = "person1", "pune", 40, "person1@test.com"
    print(f"name = {name}, type = {type(name)}")
    print(f"address = {address}, type = {type(address)}")
    print(f"age = {age}")
    print(f"email = {email}")

function1()
```

```
def function2():
    n1 = 10
    n2 = 20
    print(f"n1 = {n1}, n2 = {n2}")
    # temp = n1
    # n1 = n2
    # n2 = temp
    # print(f"n1 = {n1}, n2 = {n2}")

    # swap the values
    n1, n2 = n2, n1
    print(f"n1 = {n1}, n2 = {n2}")

function2()
```

```
def math_operations(p1: int, p2: int):
    addition = p1 + p2
    subtraction = p1 - p2
    multiplication = p1 * p2
    division = p1 / p2

    # return a tuple
    return addition, subtraction, multiplication, division


# result = math_operations(30, 10)
# print(f"result = {result}")
# print(f"addition = {result[0]}")
# print(f"subtraction = {result[1]}")
# print(f"multiplication = {result[2]}")
# print(f"division = {result[3]}")

addition, subtraction, multiplication, division = math_operations(30, 10)
print(f"addition = {addition}")
print(f"subtraction = {subtraction}")
print(f"multiplication = {multiplication}")
print(f"division = {division}")
```

multi-dimensional collection part 1

```
def function1():
    # list of numbers
    # one dimensional collection
    numbers = [10, 20, 30, 40, 50]

    # for..in loop
    for number in numbers:
        print(f"number = {number}")

    print("-" * 80)

    # traditional for loop
    for index in range(len(numbers)):
        print(f"value at {index} = {numbers[index]}")

function1()
```

```
def function2():
    # list of lists
    # rows = 3
    # columns per row = 2
    numbers = [
        [10, 20],
        [30, 40],
        [50, 60]
    ]

    # for..in loop
    for inner_list in numbers:
        for value in inner_list:
            print(f"value = {value}")
        print("")

    print("-" * 80)

    # traditional for loop
    # getting the values by index
    for row in range(len(numbers)):
        # print(f"list at {row} = {numbers[row]}")

        # get the inner list by using index position
        inner_list = numbers[row]

        for col in range(len(inner_list)):
            # print(f"value at [{row}][{col}] = {inner_list[col]}")
```

```
            print(f"value at [{row}][{col}] = {numbers[row][col]}")

function2()
```

```
def function3():
    countries = [
        ["india", "sri lanka"],
        ["Saudi Arebia", "Egypt"],
        ["Canada", "Brazil"]
    ]

    # using for..in
    for inner_list in countries:
        for country in inner_list:
            print(f"country = {country}")
        print("")

    print("-" * 80)

    # using traditional for
    for row in range(len(countries)):
        inner_list = countries[row]
        for col in range(len(inner_list)):
            # print(f"country at [{row}][{col}] = {inner_list[col]}")
            print(f"country at [{row}][{col}] = {countries[row][col]}")


function3()
```

multi-dimensional collection part 2

```
def function1():
    # list of tuples
    numbers = [
        (10, 20),
        (30, 40),
        (50, 60)
    ]

    # for..in loop
    for inner_list in numbers:
        for value in inner_list:
            print(f"value = {value}")
        print("")

    print("-" * 80)

    # traditional for loop
```

```
    # getting the values by index
    for row in range(len(numbers)):
        # print(f"list at {row} = {numbers[row]}")

        # get the inner list by using index position
        inner_list = numbers[row]

        for col in range(len(inner_list)):
            # print(f"value at [{row}][{col}] = {inner_list[col]}")
            print(f"value at [{row}][{col}] = {numbers[row][col]}")

function1()
```

```
def function2():
    # tuple
    # person1 = ("person1", 30, "person1@test.com")

    # list of tuples
    persons = [
        ("person1", 30, "person1@test.com"),
        ("person2", 33, "person2@test.com"),
        ("person3", 34, "person3@test.com"),
        ("person4", 36, "person4@test.com")
    ]

    # print(persons)
    for person in persons:
        print(f"name: {person[0]}, age: {person[1]}, email: {person[2]}")

    print(f"-" * 80)

    for person in persons:
        # unpacking tuple
        (name, age, email) = person
        print(f"name: {name}, age: {age}, email: {email}")

    print(f"-" * 80)

    # unpack every person into name, age and email
    for (name, age, email) in persons:
        print(f"name: {name}, age: {age}, email: {email}")

function2()
```

```
def function3():
    cars = [
        ('i20', 'hyundai', 14.50),
        ('fabia', 'skoda', 12),
        ('sonet', 'kia', 18),
```

```
        ('fortuner', 'toyota', 35)
    ]

    # print all the information about every car
    for (model, company, price) in cars:
        print(f"model: {model}, company: {company}, price = {price}")

function3()
```

```
def function4():
    car = ('i20', 'hyundai', 14.50)

    # print(f"model = {car[0]}")

    # unpacking car
    # not interested in second and third value (_)
    # use underscore if value is not required
    (model, _, _) = car
    print(f"model = {model}")

function4()
```

```
def function5():
    # the index position is not required in the for loop
    # so ignore it by replacing the temp variable with _
    for _ in range(5):
        print("hello world")

function5()
```

set

```
def function1():
    # list of numbers
    # insertion order and duplicate values
    numbers_list = [10, 20, 30, 40, 50, 10, 20, 30, 40, 50, 10, 20, 30, 40, 50]
    print(f"numbers_list = {numbers_list}, type = {type(numbers_list)}")

    # tuple of numbers
    # insertion order and duplicate values
    numbers_tuple = (10, 20, 30, 40, 50, 10, 20, 30, 40, 50, 10, 20, 30, 40, 50)
    print(f"numbers_tuple = {numbers_tuple}, type = {type(numbers_tuple)}")

    # set of numbers
    # does not maintain insertion order and unique values
```

```
    numbers_set = {10, 20, 30, 40, 50, 10, 20, 30, 40, 50, 10, 20, 30, 40, 50}
    print(f"numbers_set = {numbers_set}, type = {type(numbers_set)}")

function1()
```

```
def function2():
    s1 = {10, 20, 30, 40}
    s2 = {30, 40, 50, 60}

    # union of sets
    print(f"s1 union s2 = {s1.union(s2)} = {s1 | s2}")
    print(f"s2 union s1 = {s2.union(s1)} = {s1 | s2}")

    print("-" * 80)

    # intersection of sets
    print(f"s1 intersection s2 = {s1.intersection(s2)} = {s1 & s2}")
    print(f"s2 intersection s1 = {s2.intersection(s1)} = {s2 & s1}")

    print("-" * 80)

    print(f"s1 - s2 = {s1 - s2}")
    print(f"s2 - s1 = {s2 - s1}")

function2()
```

```
def function3():
    # list of students
    students = ['st1', 'st2', 'st1', 'st2', 'st3', 'st1']

    # find out the unique students
    # set is a collection of unique values so convert the list into a set
    unique_students = set(students)
    print(f"unique students = {unique_students}")

    for student in unique_students:
        print(f"student = {student}")

    # as set is unordered collection, you can not retrieve
    # values from set using position/index
    # print(f"student = {unique_students[0]}")

    # set is a collection of unique items
    # so st1 will get added only once
    unique_students.add("st1")
    unique_students.add("st1")
    unique_students.add("st1")
    unique_students.add("st1")
    unique_students.add("st5")
```

```
    unique_students.add("st4")

    print(unique_students)


function3()
```

dictionary

- collection of key-value pairs

```
def function1():
    person1 = {
        "name": "person1",
        "age": 40,
        "can_vote": True,
        "address": "pune",
        "emails": {"person1@test.com", "person1.test@gmail.com"},
        "phone_numbers": {
            "personal": "+91234224",
            "work": "+914354355"
        }

        # do not use same key multiple times
        # "address": "mumbai"
    }

    print(person1)
    print(f"type = {type(person1)}")
    print(f"keys = {person1.keys()}")
    print(f"values = {person1.values()}")

    # person1['name'] will return value of key name
    print(f"name = {person1['name']}")
    print(f"age = {person1['age']}")
    print(f"can vote = {person1['can_vote']}")
    print(f"address = {person1['address']}")
    print(f"emails = {person1['emails']}")

    print(f"phone numbers = {person1['phone_numbers']}")
    print(f"phone number (personal) = {person1['phone_numbers']['personal']}")
    print(f"phone number (work) = {person1['phone_numbers']['work']}")

    # phone_numbers will refer the dictionary within person1
    phone_numbers = person1['phone_numbers']
    print(f"personal = {phone_numbers['personal']}")
    print(f"work = {phone_numbers['work']}")
    # print(f"phone_numbers keys = {phone_numbers.keys()}")
    # print(f"phone_numbers values = {phone_numbers.values()}")
```

```
function1()
```

```
def function2():
    # design a dictionary to store following details of a phone
    # model, company, price, configuration (ram, cpu), country, resolution (width,
height), screen_size
    phone = {
        "model": "iPhone 13 pro max",
        "company": "Apple",
        "price": 179000,
        "configuration": {
            "cpu": "A15",
            "ram": "8GB"
        },
        "country": "india",
        "resolution": {
            "width": 1444,
            "height": 2556
        },
        "screen_size": 7.1
    }
    print(phone)

    # get all the values for all the keys

function2()
```

```
def function3():
    # merging key and value lists
    key_list = ["name", "address", "company"]
    values_list = ["steve", "usa", "apple"]

    # empty list
    # x = []

    # empty tuple
    # y = ()

    # empty dictionary
    # z = {}

    # empty dictionary to begin with
    person = {}
    for index in range(len(key_list)):
        key = key_list[index]
        value = values_list[index]

        # add key-value pair in the dictionary dynamically
```

```
        # if key exists the value will get overwritten
        # if key does not exist then the key and value will get added in the
dictionary
        person[key] = value

    print(person)

function3()
```

from the given article find out the word counts

```
def function1():
    article = """India, officially the Republic of India (Hindi: Bhārat
Gaṇarājya),[24] is a country in South Asia. It is the seventh-largest country by
area, the second-most populous country, and the most populous democracy in the
world. Bounded by the Indian Ocean on the south, the Arabian Sea on the southwest,
and the Bay of Bengal on the southeast, it shares land borders with Pakistan to
the west;[f] China, Nepal, and Bhutan to the north; and Bangladesh and Myanmar to
the east."""

    # step 1: split the article in sentences
    # sentence: which ends with . (dot)
    sentences = article.split(".")
    # print(sentences)

    # step 2: split every sentence into words
    words = []
    for sentence in sentences:
        # separate words from sentence
        temp_words = sentence.split(" ")

        # add the words into all words collection
        # words.extend(temp_words)

        # remove the unnecessary characters before adding the words
        for word in temp_words:
            # replace unnecessary character with nothing
            # remove these characters
            processed_word = word.replace(",", "")
            processed_word = processed_word.replace("(", "")
            processed_word = processed_word.replace(")", "")
            processed_word = processed_word.replace("[", "")
            processed_word = processed_word.replace("]", "")
            processed_word = processed_word.replace(":", "")
            processed_word = processed_word.replace("%", "")
            processed_word = processed_word.replace("$", "")
            processed_word = processed_word.replace("'", "")
            processed_word = processed_word.replace(";", "")
            processed_word = processed_word.replace("-", "")

            # convert every character to lower case
```

```
            processed_word = processed_word.lower()

            # remove the spaces
            processed_word.lstrip()
            processed_word.rstrip()

            # add the processed word into the words collection
            if len(processed_word) > 0:
                words.append(processed_word)

    # step 3: count every word in a dictionary
    # {"india": 4, "country": 5 ... }
    print(f"total number of processed words: {len(words)}")

    # empty dictionary to hold word with its count
    word_count = {}
    for word in words:
        # check if the word exists in the dictionary
        # get will return the value of key exists
        # if not then it returns None
        count = word_count.get(word)
        if count == None:
            # if does not exist then set the count to 1
            # key/word does not exist
            word_count[word] = 1
        else:
            # if exists then increment the count
            word_count[word] = count + 1

    # step 4: print the result
    print(word_count)


function1()
```

```
def function1():
    # from the given article find out the word counts
    article = """India, officially the Republic of India (Hindi: Bhārat
Gaṇarājya),[24] is a country in South Asia. It is the seventh-largest country by
area, the second-most populous country, and the most populous democracy in the
world. Bounded by the Indian Ocean on the south, the Arabian Sea on the southwest,
and the Bay of Bengal on the southeast, it shares land borders with Pakistan to
the west;[f] China, Nepal, and Bhutan to the north; and Bangladesh and Myanmar to
the east."""

    # step 1: split the article in sentences
    # sentence: which ends with . (dot)
    sentences = article.split(".")
    # print(sentences)

    # step 2: split every sentence into words
```

```
words = []
for sentence in sentences:
    # separate words from sentence
    temp_words = sentence.split(" ")

    # add the words into all words collection
    # words.extend(temp_words)

    # remove the unnecessary characters before adding the words
    for word in temp_words:
        # replace unnecessary character with nothing
        # remove these characters
        processed_word = word.replace(",", "")
        processed_word = processed_word.replace("(", "")
        processed_word = processed_word.replace(")", "")
        processed_word = processed_word.replace("[", "")
        processed_word = processed_word.replace("]", "")
        processed_word = processed_word.replace(":", "")
        processed_word = processed_word.replace("%", "")
        processed_word = processed_word.replace("$", "")
        processed_word = processed_word.replace("'", "")
        processed_word = processed_word.replace(";", "")
        processed_word = processed_word.replace("-", "")

        # convert every character to lower case
        processed_word = processed_word.lower()

        # remove the spaces
        processed_word.lstrip()
        processed_word.rstrip()

        # add the processed word into the words collection
        if len(processed_word) > 0:
            words.append(processed_word)

# step 3: count every word in a dictionary
# {"india": 4, "country": 5 ... }
print(f"total number of processed words: {len(words)}")

# empty dictionary to hold word with its count
word_count = {}

# find the unique words from the words collection
unique_words = set(words)
for unique_word in unique_words:
    # set the occurance of every unique word to 0
    word_count[unique_word] = 0

for word in words:
    # just increment the count of every word found in the article
    word_count[word] += 1

# step 4: print the result
print(word_count)
```

```
function1()
```

Ques - find out if the number is prime or not

```
def is_prime():
    # to get input from user
    # value entered by user will always be a string
    str_value = input("enter a number to check if it is prime: ")

    # convert the value from string to int
    number = int(str_value)

    # consider the number as prime
    is_prime = True

    # start from 2 till (number // 2)
    for temp_number in range(2, (number // 2) + 1):
        # check if the temp_number is dividing the input number
        if number % temp_number == 0:
            # this number can not be prime
            is_prime = False
            break

    if is_prime:
        print(f"{number} is prime")
    else:
        print(f"{number} is not prime")


# infinite loop
# - the loop that never stops
while True:
    print("----------------")
    print("you options are:")
    print("(a) prime test")
    print("(e) exit")
    print("----------------")

    choice = input("enter your choice: ")
    if choice == 'a':
        is_prime()
    elif choice == 'e':
        continue
    else:
        print("wrong choice, please try again")

print("bye bye..")
```

while

- no of iterations are not known **for**
- no of iteration are known

break

- breaks the loop
- shifts the context/control to the next line after the loop
- never executes the next statements after break

continue

- continue to the first line of the loop
- like restarting the loop from first line of the loop
- never executes the next statements after continue

while

```
def function1():
    for index in range(10):
        print(f"index = {index}")

function1()
```

```
def function2():
    index = 0
    while index < 10:
        print(f"index = {index}")
        index += 1

function2()
```

```
def function3():
    # do not execute the print("after...") if break is used in the loop

    count = 0
    for index in range(10):
        print(f"index = {index}")
        count += 1
        if index == 6:
            break

    if count == 10:
        print("after for loop executed from first to the last")
    else:
        print("for loop broke in between")
```

```
function3()
```

```
def function4():
    # do not execute the print("after...") if break is used in the loop

    for index in range(10):
        print(f"index = {index}")
        if index == 6:
            print("for loop broke in between")
            break
    else:
        # this code will be executed only when the for loop consumes
        # all the iterations [iterates from first to last value]
        # if for loop breaks in between, this code will never gets called
        print("after for loop executed from first to the last")

function4()
```

```
def function5():
    str_value = input("enter a number to check if it is prime: ")
    number = int(str_value)

    for temp_number in range(2, (number // 2) + 1):
        if number % temp_number == 0:
            print(f"{number} is not prime")
            break
    else:
        print(f"{number} is prime")

function5()
```

variable length arguments function

```
def add1(p1, p2, p3=0, p4=0, p5=0):
    addition = p1 + p2 + p3 + p4 + p5
    print(f"addition = {addition}")

add1(10, 20)
```

- this is an example of variable length arguments function **-args is a parameter to get all the arguments passed to the function*

```
def add(*args):
    print("inside function add")
    print(f"args = {args}, type = {type(args)}")

    # get addition of all values from args
    # addition = 0
    # for arg in args:
    #     addition += arg
    # print(f"addition = {addition}")
    print(f"addition = {sum(args)}")


add(10, 20)
add(10, 20, 30)
```

```
def my_function(*args, **kwargs):
    print(f"inside my_function")
    print(f"args = {args}, type = {type(args)}")
    print(f"kwargs = {kwargs}, type = {type(kwargs)}")

    # try to get the operation that need to perform
    operation = kwargs.get("operation")
    if operation != None:
        if operation == "sum":
            print(f"sum = {sum(args)}")
        elif operation == "multiplication":
            multiplication = 1
            for arg in args:
                multiplication *= arg
            print(f"multiplication = {multiplication}")
        else:
            print("invalid operation")


my_function(10, 20, operation="multiplication")
my_function(10, 20, operation="sum")
```

 # Write a program that asks the user for their name and greets them with their name.

```
def function1():

    name = input("enter your name: ")
    print(f"hi {name}, good noon!!")

function1()
```

get input from user and allow only the users Alice and Bob are greeted with their names

```
def function2():
    name = input("enter your name: ")
    if (name == 'alice') or (name == 'bob'):
        print(f"hi {name}, good noon!!")
    else:
        print(f"hi {name}, not going to greet you..")

function2()
```

write a program that asks the user for a number n and prints the sum of the numbers 1 to n

```
def function3():

    str_value = input("enter a number: ")
    n = int(str_value)
    addition = 0
    for number in range(1, (n + 1)):
        addition += number

    print(f"addition = {addition}")

function3()
```

Modify the previous program such that only multiples of three or five are considered in the sum. e.g. 3, 5, 6, 9, 10, 12, 15 for n=17

```
def function4():

    str_value = input("enter a number: ")
    n = int(str_value)
    addition = 0
    for number in range(1, (n + 1)):
        if (number % 3 == 0) or (number % 5 == 0):
            addition += number

    print(f"addition = {addition}")

function4()
```

Write a program that prints a multiplication table for numbers up to 12

```
def function6():

    n = 12
    for index in range(1, 11):
        for number in range(1, n + 1):
```

```
            print(f" {index * number:4} ", end="")
        print()

function6()
```

```
def function7():
    def is_prime(number):
        for temp_number in range(2, (number // 2) + 1):
            if number % temp_number == 0:
                return False
        else:
            return True

    max_value = int(input("enter max number: "))
    prime_numbers = []
    for value in range(2, (max_value + 1)):
        # check if the value is prime
        if is_prime(value):

            # if it is prime, add it to the collection
            prime_numbers.append(value)

    print(prime_numbers)

function7()
```

string operations

```
def function1():
    str_data = "India Is mY CoUntrY"

    print(f"str_data = {str_data}")
    print(f"str_data in lower = {str_data.lower()}")
    print(f"str_data in upper = {str_data.upper()}")
    print(f"str_data in capitalized = {str_data.capitalize()}")
    print(f"str_data in casefold = {str_data.casefold()}")
    print(f"str_data in title = {str_data.title()}")
    print(f"str_data in swapcase = {str_data.swapcase()}")

function1()
```

```
def function2():
    num1 = 100
    num2 = 10000
    num3 = 100000
```

```
    # right  aligned
    print(f"| {num1:>10} |")

    # left aligned
    print(f"| {num2:<10} |")

    # center aligned
    print(f"| {num3:^10} |")


function2()
```

```
def function3():
    positive_value = 10
    negative_value = -10

    # show positive symbol
    print(f"positive value = {positive_value:+}")

    # show negative symbol
    print(f"negative value = {negative_value:-}")

    value = 10000000

    # use , as 1000 separator
    print(f"value = {value:,}")

    # use _ as 1000 separator
    print(f"value = {value:_}")

function3()
```

```
def function4():
    number = 10

    print(f"decimal = {number:d}")
    print(f"binary = {number:b}")
    print(f"octal = {number:o}")
    print(f"hexa-decimal = {number:x}")
    print(f"hexa-decimal = {number:X}")

function4()
```

```
def function5():
    value = 102323.51234
```

```
    print(f"format float: {value:0.2f}")
    print(f"exponent format: {value:e}")
    print(f"exponent format: {value:E}")

function5()
```

PoP

```
def can_vote(person):
    if person['age'] >= 18:
        print(f"{person['name']} can vote")
    else:
        print(f"{person['name']} can not vote")


def goto_office(person):
    print(f"{person['name']} is going to office")
```

```


data (attribute-value)

person1 = {
    "name": "person1",
    "address": "pune",
    "email": "person1@test.com",
    "age": 20
}

# check if person can vote

can_vote(person1)
goto_office(person1)
```

```

# data (attribute-value)
person2 = {
    "full_name": "steve jobs",
    "address": "usa",
    "email": "steve@apple.com",
    "age": 58
}

can_vote(person2)
```

oop

class will define the prototype**

```
class Person:
    """a dummy class"""
    pass
```

creating an object of Person class

```
p1 = Person()
print(f"person = {p1}, type = {type(p1)}")
```

creating another object of Person class

```
p2 = Person()
print(f"person = {p2}, type = {type(p2)}")
```

```
p3 = Person()
print(f"person = {p3}, type = {type(p3)}")
```

class: prototype of object

conventions

- class name must start with upper case
- class name must not be plural
- if class needs multiple words then use underscore to join them (School_Student)

object: instance of a class

conventions

- reference name must be all lower cased
- if reference needs multiple words then use underscore to join them

empty class

```
class Person:
    """dummy class"""
    pass
```

instantiation

- creating an object of Person class

```
p1 = Person()
```

add a new attribute inside the object

- param1: reference
- param2: attribute name
- param3: value of the attribute

```
setattr(p1, "name", "person1")
setattr(p1, "address", "pune")
setattr(p1, "age", 30)
```

get the attribute(s) from the object

```
name = getattr(p1, "name")
address = getattr(p1, "address")
age = getattr(p1, "age")

print(f"name = {name}")
print(f"address = {address}")
print(f"age = {age}")
```

create a new object

```
p2 = Person()
```

add new attributes to the object

- name, address and age are the attributes of p2
- person2, mumbai and 40 are the values of those attributes

```

```

```
setattr(p2, "name", "person2")
setattr(p2, "address", "mumbai")
setattr(p2, "age", 40)
```

get the attribute values

```
name = getattr(p2, "name")
address = getattr(p2, "address")
age = getattr(p2, "age")

print(f"name = {name}")
print(f"address = {address}")
print(f"age = {age}")
```

- create an empty class named Car
- create one object of class Car
- add attributes: model, company, price
- get all the attributes from the object

```
class Car:
    pass

# instantiation
c1 = Car()

setattr(c1, "model", "i20")
setattr(c1, "company", "hyundai")
setattr(c1, "price", 15)

print(f"model = {getattr(c1, 'model')}")
print(f"company = {getattr(c1, 'company')}")
print(f"price = {getattr(c1, 'price')}")

print("-" * 80)
```

```
# instantiation
c2 = Car()

setattr(c2, "car_model", "i10")
setattr(c2, "car_company", "hyundai")
setattr(c2, "car_price", 10)
setattr(c2, "car_color", "gray")

print(f"model = {getattr(c2, 'car_model')}")
print(f"company = {getattr(c2, 'car_company')}")
print(f"price = {getattr(c2, 'car_price')}")
```

```
print(f"color = {getattr(c2, 'car_color')}")

print("-" * 80)
```

```
# changing the attribute's value
setattr(c2, "car_color", "yellow")
# c2.car_color = "yellow"

print(f"model = {getattr(c2, 'car_model')}")
print(f"company = {getattr(c2, 'car_company')}")
print(f"price = {getattr(c2, 'car_price')}")
print(f"color = {getattr(c2, 'car_color')}")
```

```
# delete an attribute
delattr(c2, "car_color")

# this line wont work as car_color is deleted from object c2
# print(f"color = {getattr(c2, 'car_color')}")
```

```
class Person:
    # method
    def print_person_details(self):
        print(f"name = {getattr(self, 'name')}")
        print(f"address = {getattr(self, 'address')}")

    # method
    def set_attributes(self, person_name: str, person_address: str):
        # add/set attribute named name with value in the parameter person_name
        setattr(self, "name", person_name)

        # add/set attribute named address with value in the parameter
person_address
        setattr(self, "address", person_address)
```

```
p1 = Person()
# please do not uncomment this line
# this is how python will call the method internally
# Person.set_attributes(p1, person_name="person1", person_address="pune")
p1.set_attributes(person_name="person1", person_address="pune")
p1.print_person_details()

print("-" * 80)
```

```
p2 = Person()
# please do not uncomment this line
# this is how python will call the method internally
# Person.set_attributes(p2, person_name="person2", person_address="mumbai")
p2.set_attributes(person_name="person2", person_address="mumbai")
p2.print_person_details()
```

class methods part 1

```
class Person:
    def __init__(self):
        """initializer:
        - this is called as default initializer (parameterless)
        - method which gets called automatically
        - it gets called for every object of this class
        """
        print("inside __init__")

        # add default values for the required attributes
        setattr(self, "name", "")
        setattr(self, "address", "")
        setattr(self, "age", 0)

    def print_details(self):
        print(f"name = {getattr(self, 'name')}")
        print(f"address = {getattr(self, 'address')}")
        print(f"age = {getattr(self, 'age')}")

    def set_details(self, name, address, age):
        setattr(self, "name", name)
        setattr(self, "address", address)
        setattr(self, "age", age)


p1 = Person()
```

steps:

- allocates the required memory
- calls the **init** method for the object
- Person.**init**(p1)

p1's name, address and age are having default values

```
p1.set_details("person1", "pune", 20)
p1.print_details()
```

```
p2 = Person()
```

- allocates the required memory
- calls the **init** method for the object
- Person.**init**(p2)

```
p2.set_details("person2", "mumbai", 30)
p2.print_details()
```

class methods part 2

- initializer

```
class Person:
    def __init__(self, name, address, age):
        """
        - this method is called as custom initializer
        - accepts at least one parameter

        :param name: name of the person
        :param address: address of the person
        :param age: age of the person
        """
        setattr(self, "name", name)
        setattr(self, "address", address)
        setattr(self, "age", age)

    def print_details(self):
        print(f"name = {getattr(self, 'name')}")
        print(f"address = {getattr(self, 'address')}")
        print(f"age = {getattr(self, 'age')}")


p1 = Person(name="person1", address="pune", age=20)
p1.print_details()

print("-" * 80)

p2 = Person("person2", "mumbai", 30)
p2.print_details()

print("=" * 80)
```

Practice -

- create a class School
- attributes: name, address, phone
- make sure that the object gets initialized immediately after getting created

```
class School:
    def __init__(self, name, address, phone):
        setattr(self, 'name', name)
        setattr(self, 'address', address)
        setattr(self, 'phone', phone)

    def print_details(self):
        print(f"name: {getattr(self, 'name')}")
        print(f"address: {getattr(self, 'address')}")
        print(f"phone: {getattr(self, 'phone')}")


s1 = School('school 1', 'pune', '+91234232435')
s1.print_details()
```

class continue

- using self inside the class methods is mandatory
- e.g -> self.name = name
- where self. =
- every class in python will contain **init** and **del**
- if you write the methods in the class, your methods will get called
- if you dont have these methods, then python will insert them and call them by default

```
class Person:
    def __init__(self, person_name, person_address="", person_age=0):
        # initialize the object
        self.name = person_name
        self.address = person_address
        self.age = person_age

    def __del__(self):
        # like __init__ this method also gets called automatically
        # this method gets called just before the object gets deleted from the
memory
        # de-initialize: closing the files, closing the connections etc
        print(f"inside __del__ ({self.name})")

    def print_details(self):
        print(f"name = {self.name}")
        print(f"address = {self.address}")
        print(f"age = {self.age}")

    def can_vote(self):
```

```
        if self.age >= 18:
            print(f"{self.name} is eligible for voting")
        else:
            print(f"{self.name} is NOT eligible for voting")
```

```
p1 = Person("person1", "pune", 30)
# Person.__init__(p1, "person1", "pune", 30)

# Person.print_details(p1)
p1.print_details()
```

delete the object

```
del p1
```

once the object gets deleted, the object can NOT be accessed any more

```
p1.print_details()
p2 = Person("person2")
p2.print_details()
del p2
```

class - methods

accessability

- private
- wont be accessible outside of the class
- these members can be accessed only within the same class
- to declare private attributes
    - used double underscore prefix with the attribute name
    - e.g. __model, __company

- protected

- public

- accessible outside of the class
- to declare public attributes
    - do not use any underscore
    - e.g. model, company

- system attributes

- not for you to declare
- declared by python (in-built attributes)
- e.g. **init**, **del**

```
class Car:
    def __init__(self, model, company, price):
        # private
        self.__model = model
        self.__company = company
        self.__price = price

    def __del__(self):
        print("__del__ called")

    def print_details(self):
        # private members can be accessed inside the class
        print(f"model = {self.__model}")
        print(f"company = {self.__company}")
        print(f"price = {self.__price}")
```

setter method

mutator method

```
    def set_price(self, price):
        # validate the price and only then set the value
        if price >= 0:
            self.__price = price

    def set_model(self, model):
        self.__model = model

    def set_company(self, company):
        self.__company = company

    def get_model(self):
        return self.__model

    def get_company(self):
        return self.__company

    def get_price(self):
        return self.__price
```

```
# state:
# - model: i20, company: hyundai, price: 15
```

```
c1 = Car("i20", "hyundai", 15)
c1.print_details()
```

Notes -

- can not change the attribute values
- can not access the attributes outside of the class

```
c1.__model = "test model"
c1.__price = -15.50
c1.__company = "invalid company"
c1.set_price(-16)
c1.print_details()
print(f"price = {c1.get_price()}")
```

Practice -

- write a class Person
- attributes: name (private), age (private)
- methods: init, del, setters, getters and print_details

```
def function1():
    print("inside function1")

# num = 100
# print(f"num = {num}, type = {type(num)}")
print(f"function1 = {function1}, type = {type(function1)}")
```

```
class Person1:
    def __init__(self, name, address):
        # public
        self.name = name
        self.address = address

    def print_details(self):
        print(f"name = {self.name}")
        print(f"address = {self.address}")
```

```
class Person2:
    def __init__(self, name, address):
        # private
        # self._Person2__name = name
        self.__name = name
        self.__address = address
```

```
    def __print_details(self):
        print(f"name = {self.__name}")
        print(f"address = {self.__address}")
```

```
p1 = Person1("person1", "pune")

# __dict__ -> dictionary form of your object
print(f"contents = {p1.__dict__}")
print(f"name = {p1.name}")
p1.name = "changed"
print(f"name = {p1.name}")

print('-' * 80)
```

```
p2 = Person2("person2", "mumbai")

# __dict__ -> dictionary form of your object
print(f"contents = {p2.__dict__}")
# p2.__name = "changed"

# do not write such code.
# because may be in the future versions, the name mangling changes its pattern
# p2._Person2__name = "chagned"
print(f"contents = {p2.__dict__}")
print(f"contents = {Person2.__dict__}")
# print(f"name = {p2.__name}")
# p2.__print_details()
```

- do not write such code.
- because may be in the future versions, the name mangling changes its pattern
- p2._Person2__print_details()

Name Mangling

- behind the scene the private attribute name gets changed
- to ___
- e.g. __name => _Person2__name