Module 13

Functions

Function is a user defined piece of code. It works only when it is called by user. It helps in reusability of code and helps in reducing errors in your code. Example: print, range, etc.,

- Helps in reusability of code
- Make code manageable and organized
- Functions are of two types:
  - User defined
  - Built-in

```
# Defining a function
def greet():
    print("Hey, have a nice day!!")
type(greet)
<class 'function'>
print(greet)
<function greet at 0x7fe3498d0670>

# calling a function
greet()
Hey, have a nice day!!
```

Doc String

```
greet?
Signature: greet()
Docstring: <no docstring>
File: /var/folders/2n/...../41735549618.py
Type: function
def greet():
    """
    This function greets everyone when it is called.
    """
    print("Hey, have a nice day!!")
```

great ?

Signature : great()

Docstring : This function greets everyone when it is called

File : /var/folders/zn/...../2375613480.py

Type : function

print ?

Docstring :

print (value, ..., sep = ' ', end = '\n', file = sys.stdout, flush = False)

Prints the values to a stream, or to sys.stdout by default.

Optional keyword arguments :

file : a file-like object (stream); defaults to the current sys.stdout.

sep : string inserted between values, default a space

end : string appended after the last value, default a newline

flush : whether to forcibly flush the stream.

Type : builtin_function_or_method.

Parameters & Arguments.

- There are place holders in the function.
- When defining them, we call them as Parameters
- When passing the actual value, we call them as Arguments

```
def great (name):          # name: Parameter
    print ("Hey, how are you!!", name)

great ("Uvaraj")           # "Uvaraj": Argument
Hey, how are you!! Uvaraj
great ("Emma")
Hey, how are you!! Emma
```

```
def add (a, b):
    c = a + b
    print (c)
add (2, 3)
5
add (2)
TypeError : add() missing 1 required positional argument : 'b'
```

Return

- A function call ends when return statement is executed.
- It returns the expression back to the function
- The code after return statement are not executed
- If there is no return value, then function returns None.

```
# Return vs Print
def add (a, b):
    c = a + b
    print (c)
add ( 3, 4)
```

```
7
c = add (3, 4)
```

```
7
type (c)
NoneType
```

← Issue !!

```
type ( print (7))
```

```
7
NoneType
```

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

b = add (3, 4)
print (b, type (b))
```

```
7   <class 'int' >
```

```
# If no return object, then it returns Nonetype
def func ():
    return

func()

c = func()
print (c, type (c))
```

```
None   <class 'NoneType' >
```

Code after return statement doesn't get executed

```
def func():
    print("Before return")
    return "Uvaraj"
    print("After return")
func()
```

Before return
'Uvaraj'

```
a = func()
```

Before return

```
a
```

'Uvaraj'

Returning multiple values

```
a, b = 2, 3
print(a, b)
```

2 3

```
def intro(name, age, hobby):
    return name, age, hobby
c, d, e = intro("Uvaraj", 35, "Travelling")
print(c, d, e)
```

Uvaraj 35 Travelling

```
f = intro("Uvaraj", 35, "Travelling")
print(f, type(f))
```

("Uvaraj", 25, "Travelling") <class 'tuple'>

When a function returns multiple values, the return type will be of form TUPLE...

Scope of a variable.

- There are two scope of a variable: Global and Local
- Global variable can be used anywhere in a program
- Local variable can only be used locally inside a program. Eg. function.

a can be used anywhere in the program

```
a = 5
def func():
    print(a)
func()
```

5

```
print(a)
```

5

```
a = 5
def func():
    x = 3
    print(x)

func()
```

3

```
print(a)
```

5

```
print(x)
```

NameError: name 'x' is not defined

```
a = 5
def func():
    a = 20
    print(a)
func()
```

20

```
print(a)
```

5

```
a = 5
def func():
    global a
    a = 20
    print(a)

print(a)
func()
print(a)
```

5
20
20

Lambda Function

There are mainly used when we need nameless functions for short period of time.

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

add (3, 4)
```

Parameters

Return value

Arguments

```
7
```

```
(lambda a, b : a + b) (3, 4)
```

```
7
```

Arguments

Return value

Parameters

```
func = lambda a, b : a + b
type (func)
function
func (3, 4)
7
```

```
def larger (a, b):
    if a > b :
        return a
    else :
        return b

larger (43, 5)
```

```
43
```

```
(lambda a, b : a if a > b else b) (43, 5)
```

```
43
```

```
large = lambda a, b : a if a > b else b
large (62, 7)
```

```
62
```

```
lst = [(12,56), (2,4), (5,3)]
lst.sort()
lst
[(2,4), (5,3), (12,56)]
```

lst.sort ?

Signature : lst.sort (*, key = None, reverse = False)

Docstring : Sort the list in ascending order and return None.
The sort is in-place (ie. the list itself is modified) and
stable (ie. the order of two equal elements is maintained).
If a key function is given, apply it once to each list
item and sort them, ascending or descending, according
to their function values.
The reverse flag can be set to sort in descending order.
Type : builtin_function_or_method

```
def k(x):
    return x[1]
lst.sort(key = k)
lst
[(5,3), (2,4), (12,56)]
```

```
lst = [(12,56), (2,4), (5,3)]
lst
[(12,56), (2,4), (5,3)]
lst.sort(key = lambda x : x[1])
lst
[(5,3), (2,4), (12,56)]
```

Challenges

① Write a Python function to print the even numbers from a given list.

- input = [1, 2, 4, 3, 5, 6]
- Output = 2 4 6

```
def even (li):
    # Iterate on the list li
    for i in li:
        # check for even elements
        if i % 2 == 0:
            print (i, end = " ")

lst = [1, 2, 4, 3, 5, 6]
even (lst)
```

2 4 6

② Write a Python function that takes a list and returns a new list with unique elements of the first list.

- input = [1, 2, 3, 1, 2, 4]
- Output = [1, 2, 3, 4]

```
lst = [1, 2, 3, 1, 2, 4]
12 in lst
```

False

```
3 in lst
```

True

```
def unique (li):
    new = []
    # Iterate on the li list
    for i in li:
        # Adding only unique elements in new list
        if i not in new:
            new.append (i)
    # Return new list
    return new

unique (lst)
```

[1, 2, 3, 4]

Types of Arguments

① Positional Arguments

- The value you pass when calling a function are matched according to their positions.

```
def intro(name, hobby):
    print("Hey, My name is ", name)
    Print("And, My hobby is ", hobby)

intro("Uvaraj")
```

TypeError: intro() missing 1 required positional argument: 'hobby'

```
intro("Uvaraj", "Travelling")
```

Hey, My name is Uvaraj
And, My hobby is Travelling

```
intro("Travelling", "Uvaraj")
```

Hey, My name is Travelling
And, My hobby is Uvaraj

② Default Arguments

- Giving default values to the parameters
- For these parameters, passing value in arguments is optional

```
def intro(name, hobby = "Travelling")
    print("Hey, My name is", name)
    print("And, My hobby is", hobby)

intro("Uvaraj")
```

Hey, My name is Uvaraj
And, My hobby is Travelling

```
intro("Emma", "Swimming")
```

Hey, My name is Emma
And, My hobby is Swimming

Default Argument is over written.

```
print ?
Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout,
      flush=False)
```

Default Arguments

Positioning of default and non-default arguments

```
def intro (name = "Uvaraj", hobby)
    print ("Hey, My name is ", name)
    Print ("And, My hobby is ", hobby)
```

✗

SyntaxError : non-default argument follows default argument

```
def intro (name, hobby = "Travelling", age)
    print ("Hey, My name is ", name)
    print ("My hobby is ", hobby)
    Print ("My age is ", age)
```

✗

SyntaxError : non-default argument follows default argument

```
def intro (name, age = 35, hobby = "Travelling")
    Print ("Hey, My name is ", name)
    Print ("My Age is ", age)
    print ("My hobby is ", hobby)
```

✓

```
intro ("Uvaraj")
```

Hey, My name is Uvaraj
My Age is 35
My hobby is Travelling

Default Arguments should always followed by the non-default Arguments.

③ Arbitrary Arguments

- When number of values you want to pass is not known
- Like we pass multiple values in print function
- The values are being stored in Tuple

```
print ?
Docstring:
print (value, ..., sep = ' ', end = '\n', file = sys.stdout,
                                          flush = False)

print (2)
2
print (2, 3)
2, 3
print ()

-
```

```
def test (*args)
    print(args)
test
()
test(2)
(2,)
test(2,3)
(2,3)
```

```
def test (*args)
    print(args)
    print (type(args))
test(2,3,4)
(2,3,4)
<class 'tuple'>
test(2,3,4,"Uvaraj",12.3)
(2,3,4,'Uvaraj',12.3)
<class 'tuple'>
```

```
def test (*args)
    print(type(args))
    # iterate and print
    for i in args:
        print(i, end=" ")
test(2,3,4,"Uvaraj",12.3)
<class 'tuple'>
2 3 4 Uvaraj 12.3
```

```
def test (*args)
    print(type(args))
    for i in args:
        print(i*i, end=" ")
test(2,3,4,5,6)
<class 'tuple'>
4 9 16 25 36
```

④ Keyword Arguments

- Variable number of key word arguments
- It stores the data in Dictionary format.

```
def intro (**kwargs):
    print (type (kwargs))
    print (kwargs)

intro ()
<class 'dict'>
{}
intro (name = "Uvaraj", age = 25)
<class 'dict'>
{'name': 'Uvaraj', 'age': 25}
```

```
def intro (**kwargs):
    for key, values in kwargs.items ():
        print (key, values, sep = " : ")
intro (name = "Uvaraj", age = 25)
name : Uvaraj
age : 25
intro (name = "Uvaraj", age = 25, hobbies = ["Swim", "Read", "cycle"])
name : Uvaraj
age : 25
hobbies : ['Swim', 'Read', 'Cycle']
```

```
def mix (a, b, c, age = 25, *args, **kwargs):
    print (a, b, c)
    print (age)
    print (args)
    print (kwargs)
mix (2, 4, 5)
2 4 5
25
()
{}
```

```
mix(2,4,5,45,6,8,9, name="Uvaraj", hobby="Swimming")
2 4 5
45
(6, 8, 9)
{'name': 'Uvaraj', 'hobby': 'swimming'}
```

```
def intro(**kwargs, *args):
    print(args)
    print(kwargs)
```

SyntaxError: invalid syntax

```
def intro(*args, **kwargs)
    print(args)
    print(kwargs)
```

✓ Keyword Argument should always follow the Arbitrary Argument.

Challenges

① Find the area of circle.

Write a function to calculate and return the area of a circle by using the radius of the circle given as a parameter.

Notes:

1. Round up the area to 2 decimal places using round() function
2. Use pi as 3.14159
3. You need not take input in this problem, you need to only implement the function provided.

Input format:

The first line indicates the number of Test cases. For each TC, there will be one line of input in integer format representing radius.

Output format:

Area in float format rounded upto 2 decimal places for each Test case.

Sample Input	Sample output
1 5	78.54

Sample Explanation

The area for circle with radius 5cm is

$3.14159 \times (5 \times 5) = 78.53975\ cm^2$

After rounding by 2 digits, it becomes $78.54\ cm^2$.

```
def Circle_area (r)
    ans = None
    pi = 3.14159
    ans = round (pi * r * r, 2)
    return ans

Circle_area (5)
78.54
```

② Time to end CORONA.

Given three integers A, B and C. You have to find the number of days it will take to reach zero cases of corona in a city.

A - Average cases recovered in a day of the corona

B - No. of new cases of corona daily

C - Current active cases of the corona

Return the minimum number of days it will take to reach 0 active cases of Covid.

Problem constraints

$1 <= B < A <= 5000$

$1 <= C <= 100$

Input format

The first Argument will be integer A, which denotes the recovered cases in a day. The second argument will be integer B, which denotes the new cases in a day. The third argument will be integer C, which denotes the currently active cases.

Output Format

Return an integer which denotes the minimum days to reach 0 cases.

Sample Input / output

Input 1 :

A = 5
B = 3
C = 1

output 1 :

1

Input 2 :

A = 4
B = 3
C = 2.

output 2 :

2

Explanation 1 :

At the end of Day 1, 3 new cases of covid will arise. So, total cases at the end of Day 1 before recovery will be 1 + 3 = 4. And after recovery, there will be (4 - 5) = -1 cases, since there cannot be negative number of cases. The cases would be 0 at the end of Day 1.

Explanation 2 :

At the end of Day 1, before recovery, cases will be 2+3=5. After recovery 5 - 4 = 1 case. At the end of Day 2 before recovery, cases will be 1 + 3 = 4. After recovery 4 - 4 = 0 case.

```
class Solution:
    def solve(self, A, B, C):
        # Net decrease in cases per day
        net_decrease = A - B

        # If the net decrease is 0 or Negative, it's impossible to
        # reach 0 cases
        if net_decrease <= 0:
            return -1  # Flag for impossible cases

        # Calculate the No. of days required
        days = (C + net_decrease - 1) // net_decrease

        return days
```

③ Celcius to Fahrenheit

What is the output of the code?

```
def convert(t):
    return t*9/5 + 32
print(convert(20))
```

68.0

④ Given the following function fun1()

```
def fun1(name, age):
    print(name, age)
```

Select all the syntactically correct function calls to this function.

a) fun1("Mohit", age = 23) ✓

b) fun1(age = 23, name = "Mohit") ✓

c) fun1(name = "Mohit", 23) ✗

d) fun1(23, "Mohit") ✓

⑤ Calculate Interest

considering the Interest() function:

```
def Interest(p, c, t=2, r=0.09):
    return p*t*r
```

Which of the following function calls are legal?

a) Interest(p=1000, c=5) ✓

b) Interest(r=0.05, 5000, 3) ✗

c) Interest(5000, t=2, r=0.05) ✗

d) Interest(c=4, r=0.12, p=5000) ✓