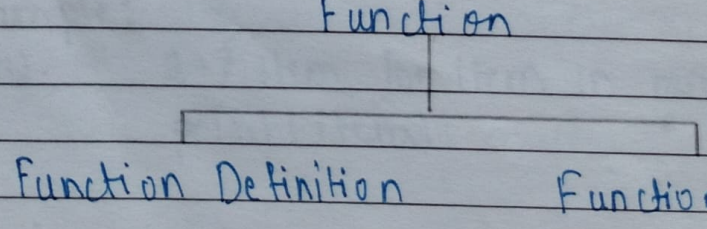
Chap-4: Functions and Modules

Chap-4A: Function

- Need of Function / What is Function
- Function Definition
- Function Call
- Variable Scope and lifetime
- The Return Statement
- Argument of function
- Lambda or Anonymous Function
- Documentation String.
- Good Programming Practices
-

Chap-4B: Modules

- Introduction to Modules
- Introduction to Pakage
- Standard Library Module



Chap-4A: Function's

▶ Need of Function:

What is Function:

Function is a block of code that perform specific task.

- following are need of Function:

1) All logically related statements can be grouped together in single entity.

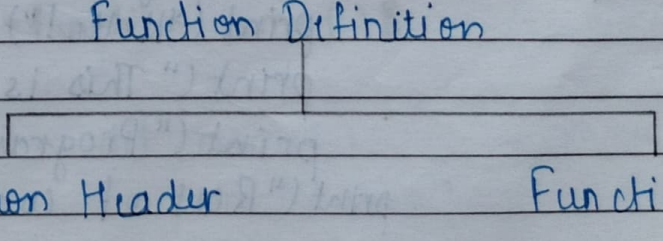
2) Use of Function makes the program easy to read, understand and debug.

3) Repeatition of frequently required code can be avoided by using function.

4) Dividing long program into function allow us to debug the part's of program one at a time.

5) Reusability is an important use/feature of Function. Once the function is declared we can use it any time anywhere, by calling it.

▶ Function Definition: it consist of two Part's



Function definition means wrapping several line of code (logically related statement) together in single entity

In Python, function is defined by 'def' keyword.

Syntax:

```
def function_name( ):  → header
    statement-1
    statement-2
    statement-3
    ----
    statement-n        } statements
rest of the code.
```

1). Every function should have function name which should be valid variable/identifier.
2). ':' colon is compulsory, it indicate that function has been started.
3) Every statement after colon are get intended to show that these are part of function.
4) Every function are associated with parameter, these parameters are enclosed within '( )' these parameter are optional, we can pass one or more parameter
5) Parameter are the input variable of function used to hold the value pass by function call

Example: #1).

```
def message( ):
    print("Hello!")
    print("This is Statement!")
    print("Program Finished.")
print("Rest of code.")
```

#2). def addition( ):
    a= 5
    b = 10
    print (a, '+', b, '=', a+b)

#3). def age( ):
    int = int(input ("Enter your age: ")
    if (age =< 18):
        print (" Child")
    else ():
        print ("Adult")
    def declaration():
        print ("Program is finished")
        print ('---')
    rest of the code

- Function Call:

Function call is a ~~funct~~ statement that invokes the function.

When a function is called, the execution control goes to the function definition and then function is get executed, after execution the control goes back to the function call. and rest of the code is get executed.

To call a function simply required to write the function name with required parameter.

Syntax: function_name()

Example: #1). def name():
    print ('Lucky')
    name()

"Every outer function can't access inner functions data-member"

#2). def message():
print (7*4)
message()

"Every inner function has access to the data-member of its prior function's."

- Variable Scope and Lifetime:

- Scope: the part/function of program in which the variable is accessible.

- Lifetime: time duration for which the variable exists.

Global Variable	Local Variable
1) A variable which is declared outside the function or in main body of Program is called as Global variable.	1) A variable which is declared inside a function is known as Local variable.
2) Accessibility of variable is throughout the program. All function can access the global variable.	2) Accessibility of variable is only within the function in which it is declared. and all the function defined within that function.
3) The global variable remains/exits through-	3) The local variable is created when the function

the entire program execution.	start executing and get destroyed when execution stops.
4) It is generally not preferred, as any function can change the values of global variable.	4) It is preferred, as local variable are more secure, these variable cannot be changed by outside function.

- **global keyword:** global key word is used to declare a global variable inside function.

The instance at which a variable is declared global, from that instance only it become global variable throughout the program.

e.g.

```
a=10  # Global
b=20  # Global
def f1():
    a=a+10
    print(a)
    print(b)
f1()
```

O/P: Error - variable 'a' referenced before assignment

```
a=10  # Global
b=20  # Global
def f1():
    global a
    a=a+10
    print(a)
    print(b)
f1()
```

O/P: a=20
b=20.

Examples:

1)
```
a=10  # Global
b=20  # Global
def f1():
    print(a,b)
    c=15  # Local
    print(c)
f1()
```

2)
```
a=100  # Global
def f1():
    def f2():
        b=200 # Local
        print(b)
    f2()
    print(100)
f1()
a=1000  # updated-G
print(a)
```

3)
```
a=100  # Global
def f1():
    def f2():
        global a # Global Statement
        a=500  # Global-updated.
    f2()
    print(a)
f1()
print(a)
```

- Return Statement: it simply assing the return value to func_name

Return statement is used to return zero or more than zero value to the calling part of function.
It is optional, it always indicate the end of function.
It is also used to transfer the control back to the calling part.

Syntax: return values

∴ if there are multiple value it returns them as a collection of tuple.

Example:

1)
```
def hello():
    msg = "hello"
    return msg
print(hello())
```

2)
```
def func():
    a, b = 5, 10
    c = a+b
    print("sum is:")
    return c
sum = func()
print(sum)
```

3)
```
def fac(n):
    if n == 0:
        return 1
    else:
        result = n * fac(n-1)
        return result
print(fac(5))
```

- **Argument of Function:**

"Argument are the variable used in function."
It is a value accepted by a function.

If a function is defined with argument, then, there is need to pass the corresponding values to calling part.

Syntax:
```
def func(arg1, arg2, arg3, ..., argn)
    ---
func(val1, val2, val3, ..., valn)
```

corresponding values get assigned to the corresponding arguments.

e.g.
```
def fun(arg):
    print(arg)
fun("Lucky")
```

- Types of argument - there are four types of argument.

1) Required Argument:

These are the necessary argument for function execution, they need to be passed in correct number and order during calling.

Syntax: def func(arg1, arg2, ...):
    ----
func(val1, value2, ...)

no. of value should be exactly equal to argument.

Example: e.g.) def add(a, b):
    c = a + b
    return c
add(5, 6)

2) def add(a, b, c)
    d = a + b + c
    print(d)
add(5, 6)
*Error - missing one required positional arg.

2) Default Argument:

These are the argument values, they are assigned to argument in function declaration part, if no. argument value is passed during function call then these values are taken as default,

Syntax: def func(arg1 = val1, arg2 = value, arg3, ...):
    ----
func(val3, value)
func(val1*, val3, ...)

Example: 1) def add(a=10, b=5):
c = a+b
add()
add(1,5)
add(20,25)

2) def add(a, b, c=20):
print([a,b,c])
add(5,10)
add(11,12,30)
add(1,1,0)

3) Keyword Argument:

Keyword argument are also refered as Named argument.
These are the argument in which argument are associated with values during calling/passing part, hence position of argument does not matter.
Each argument name should have to match with argument used in defining part.

Syntax: def func(arg1, arg2, ...):
---

func(arg2 = value, arg1 = value2, ...)
func(arg1 = value, arg2 = value2, ...)

Example: def fun(name, age):
print("name =", name)
print("age =", age)
fun(name = 'Lucky', age = 20)
fun(age = 20, nam = 'Lucky')

def add(a, b, c, d):
e = a+b+c+d
print("{0} + {1} + {2} + {3} = {4}.".format(a, b, c, d, e))
add(b=5, d=8, a=9, c=4)
add(d=5, c=4, b=2, a=1)

4) Variable length Argument:

These argument are used when we don't known what exact no. of arg. does the function accept.
It accept the argument in the form of list.

Syntax: def func(*argument_name):

# * is a special syntax for this
func(val1, val2, val3, ..., valn)

Example:
```
def fun1(*args):
    print(args) # it return the arguments in
                the form of tuple
fun1(2,4,6,8)
fun2(20,40,60)
fun1(1,2,3,....)

def sum(*lst):
    sum=0
    for i in lst:
        sum=sum+i
    return sum
print(sum(1,2,3,4,5))
print(sum(1,22,33))
```

- Lambda or Anonymous Function:

Lambda function is an anonymous function.
It is a function without a name.
It is defined by using lambda keyword.
It is defined only in single line.

Syntax: lamba arguments : expression

Example:
```
add = lambda a,b: a+b
print(add(2,5))

add = lambda a=5,b=7: a+b
print (add)

hello = lambda a: a
print(hello("hello You !!!"))

lamb = lambda : print("Hello world")
print(lamb)
```

- Documentation String / Doc string:

Doc string is just similar to comment, statement, it is a multiline string used to explain the function.

Rather than hiding like comment, it can be displayed, using --doc-- method.

Syntax:
```
def fun(x,y):
    """
    doc_string
    """
    ---
print(fun.__doc__)
```

Example:
```
def fun1():
    "this is doc-string,
     it is multiline"
    print("hello You")
```

```
def fun_name(a,b):
    """Program for addition of two no.
       a & b are passed as argument
       c = a+b is returned"""
    return a+b
```

- Good Programming Practices:

1) Make use of Meaning Full Variables.

2) Insert blank line to separate function, classes and different programming constraint.

3) Use appropriate comment in the program.

4) Make use of document string to explain more about the function used.

5) Use spaces around the operators.

6) Function name should be written in lower-case, make use of underscore '_' to seperate names.

7) The class name should be written as first letter capital.

8) Save each-and every program that you write.

9) Give the name to file, as it describe the program -

Chap- 4B: Modules

• Introduction to Modules:

Basically module in python are .py files in which set of functions are written.
Modules are imported/used using 'import' command.

• The 'from ... import' statement:

There are some standard functionalities in python that are written in particular modules. For using these functionalities, it is essential to import the corresponding module.

1) We can simply use 'import' keyword to use particular module, by using only 'import' we can use any variable and function of module.

e.g. import math
math.sqrt(25)
>>> 5.0

here, 'math' is a module, we are using sqrt (square-root) function of it.

Syntax: import module-name
module-name (value / Expression / arguments)

2) We can also use 'from --- import' statement, but we can only use selected variable or function present within that module.

e.g.
```
from math import sqrt
print('Square root(25)=', sqrt(25))
```

Here we have imported math module, but we can use only sqrt (square-root) function of module.

Syntax:
```
from mod-name import fun_name
fun_name(argument/Expression)
```

- How to Create a Module:

We can create our own module. Every python Program is a module.

Every file with .Py extension is a module, the file name is used to import particular module.

- Step-1: Creat a python file, create a function in it and save it.

e.g.
```
def add(a,b):
    print(a+b)
```

Save this file with mod1.py

Step-2: create another python file, import the module created previously by using file name and use the function.

e.g. # another python file

```
import mod1          # module imported
mod1.add(10,5)       # using function
                       of module.
```

- The dir() function:

It is used to display functions, variables, classes used in a particular module.

e.g.
```
def msg():
    print("Hello You")
name='Lucky'
print(dir())          # save this file as module-1
```

it will return all the attribute of module-1.

- Name space:

Everything in python is simply a object, Name (also called as Identifiers) is a way to access the underlying object.

e.g. a=2
2 is an object
'a' is a name given to '2'

id() function is used to get the address (in RAM) of objects.

Namespace is basically a collection of different names.
You can imagine a namespace as mapping of every name used for that objects.

In Python each module ~~same~~ own name, this namespace includes names of its function and variable.

- Global, Local and Built-in Namespace:

The global namespace contain the module which is currently executing.

The local namespace is a namespace for defining the names in a local function

The Built-in namespace is a namespace in which built-in functionality can be invoked.

eg.
```
import math
def even_num(num):    # global namespace.
    number = num      # local namespace
    if(num % 2 ==0)
        return num
    else:
        return 0
print("Enter some number")
num = int(input())  # global namespace
if(even_num(num) != 0):
    print("The square root of", num, "is", math.sqrt(num))
```
builtin namespace

- Private Variable:

The variables that begin with two underscore (__) are called as private variables.

These are the variable which are used only within that module, other module cannot access it.

e.g.
```
# creating a Module
def add(a,b):
    c = a+b
    print(f"{a} + {b} = {c}")
aa = 253
__bb = 342
# save the file as Module-1
```

variable aa is a public variable, it can be accessed by other module also.

variable __bb is a private to module it can't be accessed by other modules.

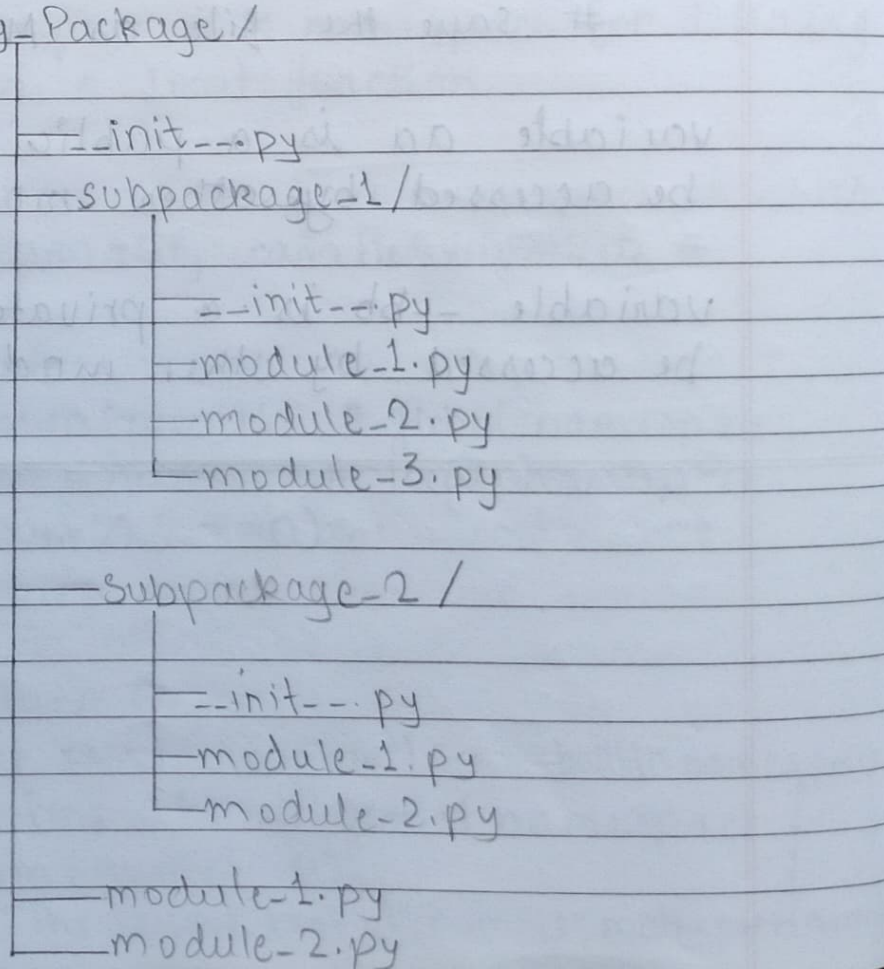
__init__.py is used to initialize python Package.

- Introduction to Packages:

Generally Packages are the folders containing certain files.

Packages are the namespaces which contains multiple sub packages and modules.
Along with sub-packages and modules, it contains other python file and __init__.py file.

For a folder, to be a package, compulsorily there should be one __init__.py file in every package and sub-package.



we can access the package and various subpackages and modules in it as follows:

e.g. import My-Package
import My-Package.module-1
From My-Package import module-2
import My-Package.Subpackage-1

* When we import Main Package then there is no need to import subpackage.

- How to Create a Package:

Creating a Calculator for performing basic operations.

Step-1: Create a folder named MyMaths. in your working folder.

Step-2: Inside MyMaths directory create subdirectories and __init__.py file i.e. an empty file.

Step-3: Inside MyMaths create a sub-directory as Add inside Add create __init__.py file and a module named as addition, in addition module create add() function as,

MyMath
init
Add
init
addition
def add

```
def add(a,b):
    c = a + b
    print(f"{a} + {b} = {c}")
```

Step-4: Similarly, inside MyMaths create another sub-directory as sub; now inside sub create --init--.py file and a module named as subtraction.py in subtraction.py module create sub() function as,

```
def sub(a,b):
    c= a-b
    print(f"{a} - {b} = {c}")
```

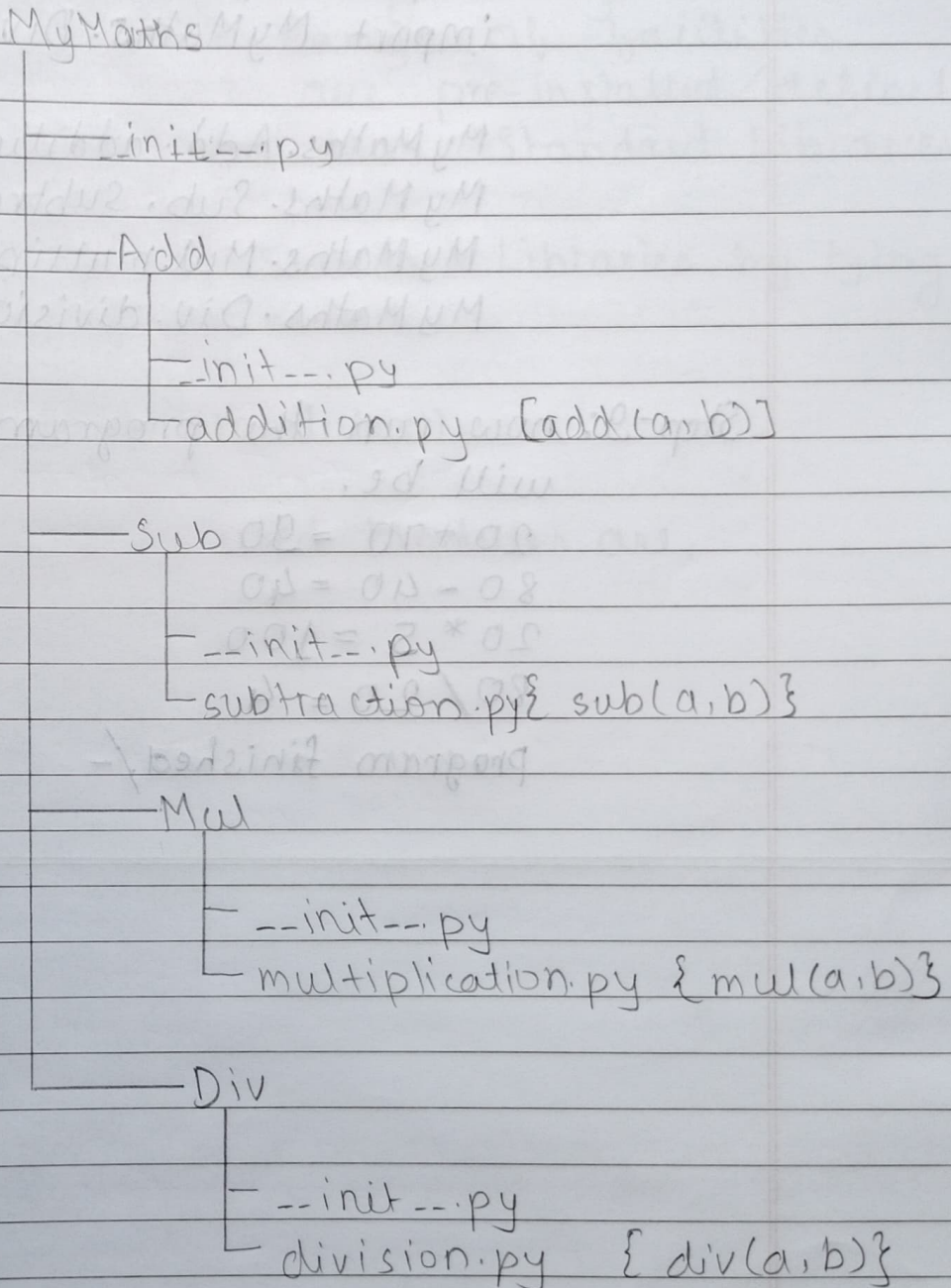
Step-5: Similarly, inside MyMaths create a sub directory named as Multiplication inside multiplication create --init--.py file and a module multi.py and inside multi.py create function mul(a, b) as

```
def mul(a,b):
    c= a*b
    print(f"{a} * {b} = {c}")
```

Step-6: Similarly create Div folder in MyMath, in Div create --init--.py and division module in it define div(a,b) function

```
def div(a,b):
    c= a/b
    print(f"{a} / {b} = {c}")
```

Step-7: The overall structure of directory will be as,



Step-8: Now, create a diriver program that invoke all the above functionali-ties present in MyMath. package.

Step-8: import MyMaths.Add.addition
import MyMaths.Sub.subtraction
import MyMaths.Mul.multiplication
import MyMaths.Div.division

MyMaths.Add.addition.add(20,70)
MyMaths.Sub.Subtraction.sub(80,40)
MyMaths.Mul.multipⁿ.mul(20,5)
MyMaths.Div.division.div(80,20)

Step-9: now run the program and output will be.

20+70 = 90
80-40 = 40
20*5 = 100
80/20 = 4
program finished/-

Introduction to Standard Libraries:

Python's Standard library is very extensive, offering a wide range of facilities The module that are pre-installed / defined in Python are called as 'Standard Libraries'.

We can get list of Standard Libraries by tying the following command,

```
>>> help('modules')
```

Some commonly used Modules are,

- time
- sys
- os
- math
- email
- random
- string
- urllib
- re
- cgi
- socket

The purpose of standard library modules is to encourage and enhance the prortability of python programs.