Functions

→ In programming, the use of function is one of the means to achieve modularity & reuseability. Functions can be defined as a named group of instructions that accomplish a specific task when it is invoked.

→ A function can be called repeatedly from different places of the program without writing all the code of that function.

Defining a function

(i) In python fun^n is defined using the def keyword

Ex
```
def printHello():
    print("Hello")
printHello()
```
Master code

Driver code

o/p
Hello

→ calling a fun^n / Invokation of fun^n

Note: Fun^n calling / Fun^n Invokation → Driver code
Fun^n Definition → Master code

(ii) Any number of parameters can be placed in the fun^n definition parenthesis.

Ex
```
def add(x, y):
    return x+y
```

Here x & y are the parameters of 'add' fun^n

Advantages of Function

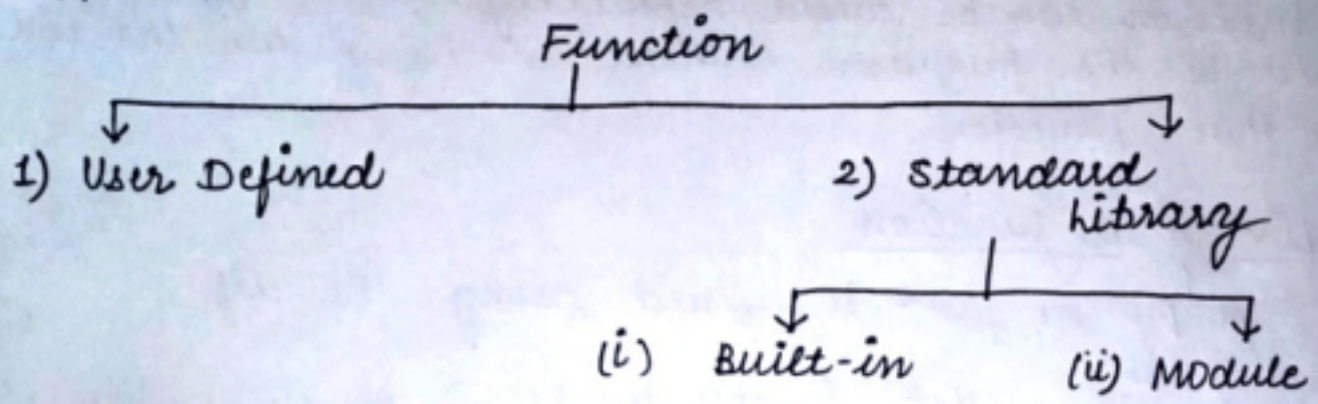
There are the following advantages of using functions in a program -:

1) Increases readability, particularly for longer code as by using functions.
2) Reduces code length as same code is not required to be written at multiple places in a program.

3) Increases reusability, as fun^n can be called from another function.

4) Work can be easily divided among team members and completed in parallel.

Types of Functions in Python



1) User Defined Functions

We can create our own function based on our requirements.

Creating user-defined fun^n

Syntax

Function Header { def <Function_name> ([parameters, ...]):

Function Body { set of instructions
[return <value>]

- items enclosed in "[ ]" are called parameters and they are optional.

(i) Fun^n may or may not have parameters.
(ii) Fun^n may or may not have return value

- Header always enclosed with (:) colons.
- Rules of defining a identifier also applied on fun^n name

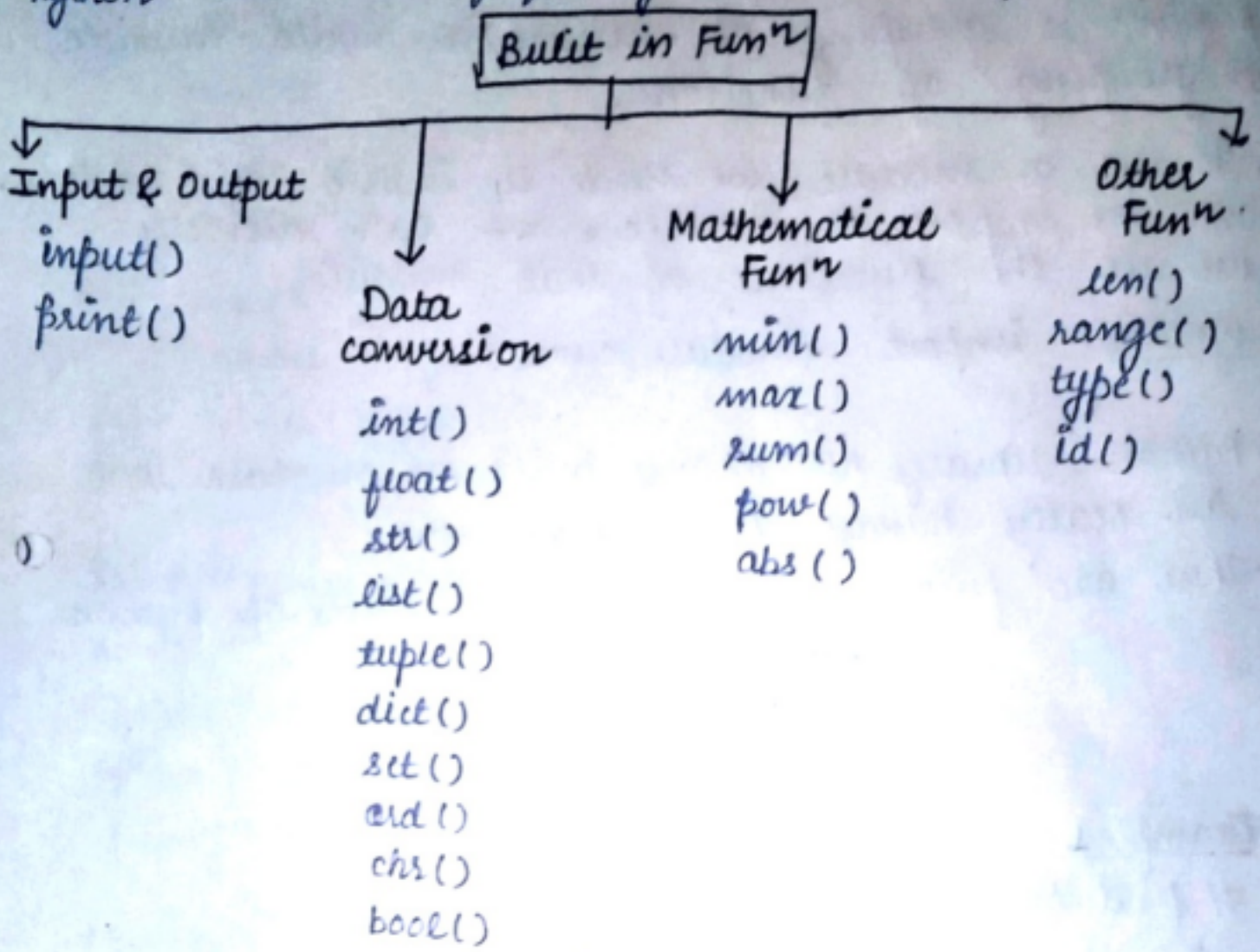
Example

```
def sub(x,y):
    return x-y
print(sub(5,4))
```

O/P
1

2(i) **Built in Functions**

Built-in fun^n are the ready-made functions in Python that are frequently used in programs.



commonly used Built-in fun^n

Fun^n syntax	Description	Example

abs(x)	returns absolute value of x	>>> abs(-8.4)
		8.4
		>>> abs(9)
		9

min(sequence) max(sequence) sum(numeric sequence)	sequence (list, tuple, string)	>>> min("punctual") a # based on ASCII valu
pow(x, y)	x^y (x raised to power y)	>>> pow(5,2) 25.0

2(ii) Module

- Other than built-in functions, the Python standard library also consist of a number of modules.
- A fun^n is grouping of instruction, while module is grouping of functions.
- To use a module, we need to import the module. Once we import a module, we can directly use all the functions of that module.

syntax import <module_name>

- Python library has many built-in module that are really handy to programmers.
- There are few commonly used modules of python
  - math
  - random
  - statistics

Example 1

```
# find square root of a number
import math
n = int(input("Enter any number : "))
print(math.sqrt(n))
```

o/p Enter any number : 9
3.0

Example 2

```
# find mean of a sequence
import statistics
l1 = [10, 20, 30, 40, 50, 60]
print(statistics.mean(l1))
```

o/p 35

Example

Calculator using fun^n

```
def add(x, y):
    return x+y
def sub(x, y):
    return x-y
def mul(x, y):
    return x*y
def divide(x, y):
    return x/y
```

```
print("Please select your choice :")
print("1 Add")
print("2 Subtract")
print("3 Multiply")
print("4 Divide")

choice = int(input("Please enter your choice:"))
num1 = int(input("Please enter first number :"))
num2 = int(input("Please enter second number :"))

if choice == 1:
    print(num1, "+", num2, "=", add(num1,num2))
elif choice == 2:
    print(num1, "-", num2, "=", sub(num1,num2))
elif choice == 3:
    print(num1, "*", num2, "=", mul(num1,num2))
elif choice == 4:
    print(num1,"/", num2, "=", divide(num1,num2))
else:
    print("Please enter valid choice")
```

Output

```
Please select your choice :
1 Add
2 Subtract
3 Multiply
4 Divide
Please enter your choice: 2
Please enter first number : 5
Please enter second number : 2
5 - 2 = 3
```

Lambda Functions / Anonymous Function

- A python lambda fun^n is simply an anonymous fun^n. It could also be called as "nameless" fun^n.
- Normal python fun^n are defined by the **def** keyword. Lambda fun^n in python usually composed of the **lambda** keyword.

Syntax

lambda argument(s) : expression

Here, argument(s) : any value passed to the lambda fun^n
expression : expression is executed & returned

Example

```
# lambda fun^n named as greet
greet = lambda : print("Hey ! I am lambda fun^n")
# call the lambda
greet()
```

O/P Hey! I am lambda fun^n

Example

```
# lambda fun^n with argument
greet = lambda subjectName : print("Hey ! You are learning", subjectName)
greet("PYTHON")
```

O/P Hey! You are learning PYTHON

Higher Order Functions in Python

A function is called Higher Order Function if it contains other function as a parameter or returns a function as an output.

Properties

- A fun^n is an instance of the object type
- We can store a fun^n in variable.
- We can pass a fun^n as a parameter.
- We can return a fun^n from a function.

Example

```
def fun1(fun2):
    return fun2

def fun2(x):
    print(x)

fun2 = fun1(fun2)
fun2(1)
```

O/P 1

```
#another way
fun1(fun2)(1)
```

O/P 1

Parameters

A parameter is the variable defined within the parenthesis during fun^n definition.

Ex

```
# Here a & b are parameters
def sum(a,b):
    print(a+b)
sum(2,3)
```

O/P 5

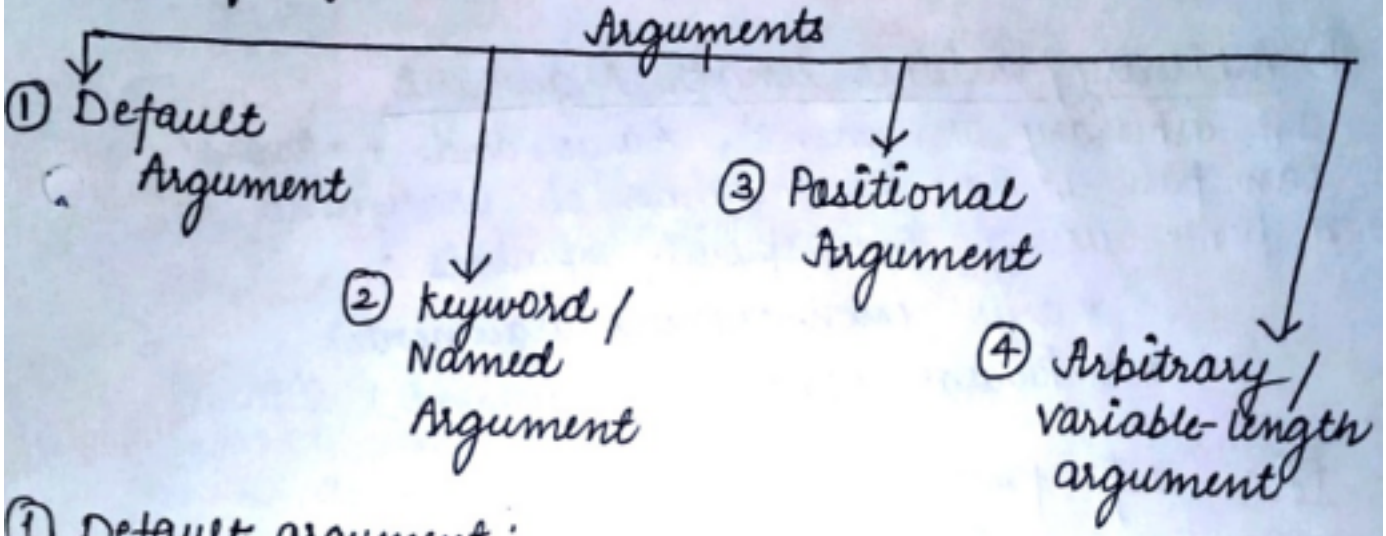
Arguments & Parameters

Arguments are the values passed inside the parenthesis of fun^n call / fun^n invokation.
A fun^n can have any number of arguments seperated by comma.

Example
```
def add(x,y):
    return x+y
print(add(5,3))
```
O/P: 8

→ Here 5,3 are arguments.

Types of arguments



① Default argument:

A default argument is a parameter that assumes a default value if a value is not provided in the fun^n call / fun^n invokation.

default argument

```
def calculatePow(x, y=2):
    print(x**y)

calculatePow(10)
```
O/P: 100

- all the arguments after default argument must also have default values.

② Keyword Arguments:

This method allows us to specify argument name with value so that we don't have to remember the order of parameters.

Ex
```
def add(x,y):
    print(x+y)
add(y=7, x=2)
```
O/P: 9

③ **Positional Arguments**

In this method, we have to remember order of parameters of fun^n definition, to assign argument value to specified parameter.

Example-

```
def employeeData(name, age):
    print(name, " ", age)
employeeData('ABC', 12)
```

O/P ABC 12

④ **Arbitrary / Variable length Argument**

In arbitrary argument, *args and **kwargs can pass a variable number of argument to a fun^n using two special symbols :

*args (non-keyword argument)
**kwargs (keyword argument)

Ex
```
def fun(*args, **kwargs):
    for arg in args:
        print(arg)

    for key,value in kwargs:
        print(key, "=", value)

fun('Hello', 'Welcome', 'to','the', 'world','o
    , sub = 'Python Programming')
```

O/P
```
Hello!
Welcome
to
the
world
of
Python Programming
```

Scope of a Variable

- A variable defined inside a function cannot be accessed outside it.
- Every variable has a well-defined accessibility. The part of the program where a variable is accessible can be defined as the scope of that variable.
- A variable can have two scopes -

Variable scope
- Global scope
- Local scope

(i) A variable that has global scope is known as Global Variable.

(ii) A variable that has local scope is known as local variable.

(A) Global Variable

- A variable that is defined outside a fun^n or any block is known as a global variable.
- It can be accessed in any fun^n defined onwards.
- Any change made in global variable impact all the fun^n in the program.

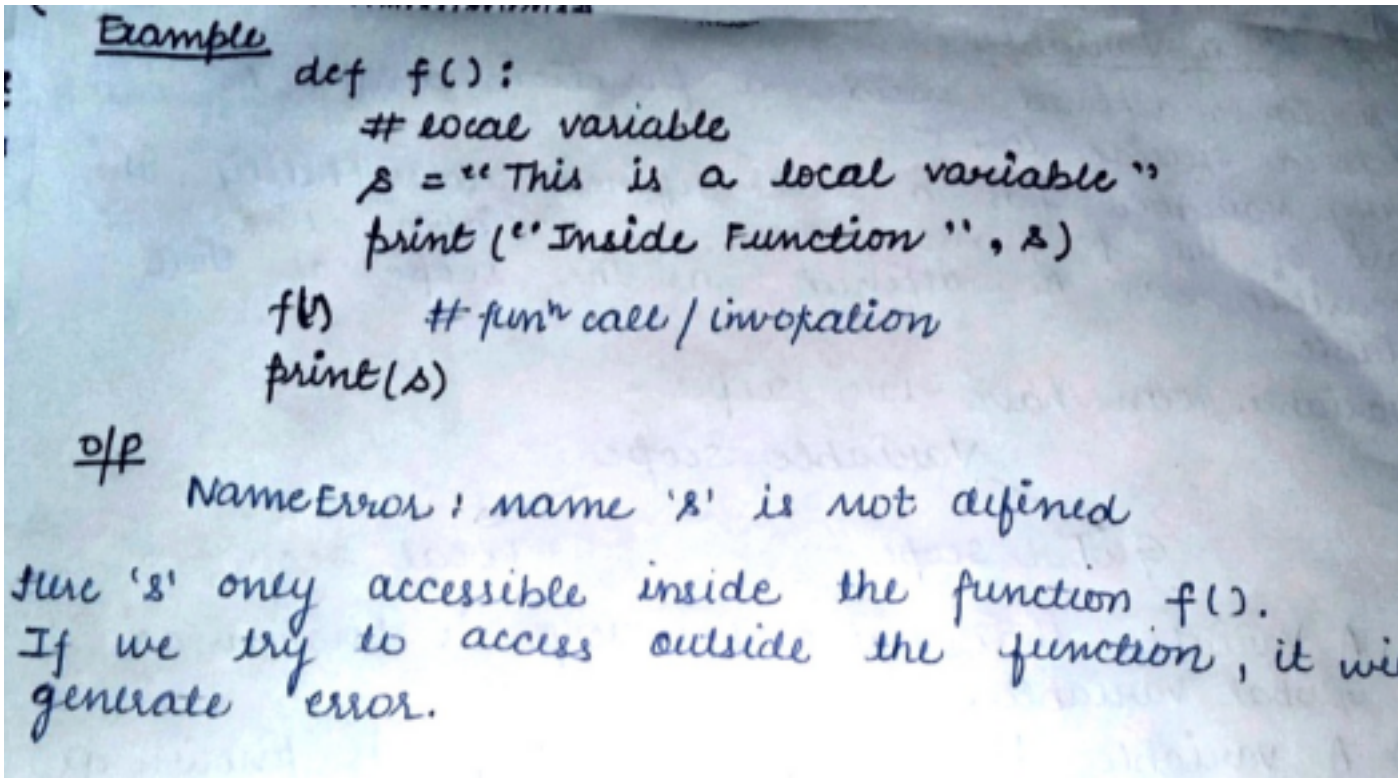
Example

```
def f():
    print("Inside Function",s)
s = "This is a global variable"    #global variable
f()    # fun^n call / invocation
print("Outside Function", s)
```

o/p
```
Inside Function This is a global variable
Outside Function This is a global variable
```

(B) Local Variable

- A variable that is defined inside any fun^n or a block is known as a local variable.
- It can be accessed only in the fun^n or a block where it is defined.



Example

```
def f():
    # local variable
    s = "This is a local variable"
    print("Inside Function", s)
f()    # fun^n call / invocation
print(s)
```

o/p

NameError : name 's' is not defined

Here 's' only accessible inside the function f().
If we try to access outside the function, it will generate error.