Chapter - 9
Functions

Functions are reusable blocks of code that perform a specific task. They help in organizing code, reducing redundancy, and improving readability and maintainability.

Need for functions:-

- Reusability:- write once, use many times.
- Modularity:- Breakdown complex problem into simple pieces.
- Maintainability:- Easier to update and manage code.
- Readability:- Improve the clarity of the code.

Types of functions:-

1. Built - in functions:- functions provided by python. (eg:- print(), len(), range())
2. user - defined function:- function defined by user to perform specific task.
3. Anonymous functions:- functions defined using the 'lambda' keywords.

Defining and calling functions

Defining a function

```
def function_name(parameters):
    """docstring (optional): Describe the function"""
    # function body
    return value    # optional.
```

Ex-

```
def greet(name):
    """Greet a person by name"""
    return f"Hello, {name}!"
print(greet("Alice"))
```

Calling a function

```
result = function_name(arguments).
```

Ex-

```
print(greet("Sagar"))
```

* Function Arguments.

1. Positional Arguments:— Arguments passed in a specific order.
2. Keyword Arguments:— Argument passed by explicity naming the parameter.

3. Default argument:- parameter with default value.

4. Arbitary Arguments:- Accepts variable number of argument using '*args" and '**kwargs'.

Example

```
def example_func(a, b, c=3, *args, **kwargs):
    print(a,b,c)
    print(args)
    print(kwargs)

example_func(1,2)
example_func(1,2,4,5,6, e=7, f=8)
```

~~Example:-~~

~~def example_func(a,b,c=3 *args *~~

Return and yield:-

- Return:- Exits the function and optionally passes back a value to the caller.

```
def add(a,b):
    return(a+b)
print(add(2,3))
```

DATE
PAGE NO.

• yield :- Used in generators to return a value and pause the function, maintaining its state.

```
def generate_numbers():
    for i in range(5):
        yield i

gen = generate_numbers()
print(next(gen))
print(next(gen))
```

* 'None' Keyword
None is used to signify the absence of a value.

```
def do_nothing():
    pass

result = do_nothing()
print(result)
```

* Scope of variables:

• Local Scope :- variable defined within a function.

• Global Scope :- variable defined outside all function.

• Non local Scope :- variable defined in the nearest

enclosing scope (excluding global scope)

Example :-

```
x = "global"
def outer ()
    x = "outer"
    def inner ():
        nonlocal x
        x = "inner"
        print ("Inner:", x
    inner ()
    print ("outer:", x)
outer ()
print ("Global:", x)
```

* Recursion

A function calling itself to solve a smaller instance of the same problem.

```
def factorial (n):
    if n == 1
        return 1.
    else:
        return n * factorial (n-1)
print (factorial (5))
```

* Anonymous functions :-
Defined using the 'lambda' Keyword.

square = lambda x : x**2
print(square(5))

# lambda function with map
nums = [1,2,3,21]
squared_nums = list(map(lambda x: x**2, nums))
print(squared_nums)

* ~~Examples :-~~

Summary :-

- Function are reusable blocks of code that perform specified task.
- Arguments can be positional, keyword, default or arbitary
- Return exits the function and returns a value, while yield is used in generators.
- Scope defines the visibility of variables (local, global, nonlocal)
- Recursion involves a function calling itself.
- Anonymous function are defined using the 'lambda' Keywords.

Ch-10

Modules and packages

Why modules?

Modules are used to break down large programs into smaller, manageable and organised files.

They help in:

- Code Reusablity:— functions and classes can be reused in different Programs.
- Maintainability: Easier to update and manage code.
- Namespace management:— Avoids name conflicts by encapsulating function, classes and variables.
- Code organization:— makes the codebase more structured and easier to navigate.

* Module creation:—

A module creation a python file (`.py`) that contains function, classes, and variables.

Example

```
def greet(name):
    return f"Hello, {name}!"

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

Importing modules:-
You can import module using 'import' statement.

```
import mymodule
print (mymodule.greet ("Alice"))
print (mymodule.add (2,3))
```

* you can also import specific functions or variable from a module:-

```
from mymodule import greet, add
print (greet ("bob"))
print (add (4,5))
```

Or import all names from a module (not recommended) due to potential name conflicts.

```
from mymodule import *
print (greet ("Satyam"))
print (add (6,7))
```

* module Namespace:
Each module has its own namespace, which helps in avoiding name conflicts.

```
# mymodule.py
    x = 10
    def show()
        print(x)
# another_module.py
    x = 20
    def display():
        print(x)
# main.py
    import mymodule
    import another_module
    mymodule.show()
    another_module.display()
```

* A namespace is a container where names are mapped to objects. In python, each module creates its own namespace, which means that variables function and classes defined in one module do not interface with these defined in another module. This helps avoid name conflicts, especially when multiple modules are imported into a single script.

* Package : Basics
A package is a collection of related modules stored in a directory that includes a special '__init__.py' file.

Creating a package

Directory Structure:
mypackage /
__init__.py
module1.py
module2.py

'__init__.py' can be an empty file but can execute initialization code for the package.

Examples:
```
# module1.py
def func1():
    return "This is function 1"

# module2.py
def func2():
    return "This is function 2"
# __init__.py
from .module1 import func1
from .module2 import func2
```

* Importing a package:-
```
import mypackage
print(mypackage.func1())
print(mypackage.func2())
```

Path Setting :-
To import a module from a different directory you can modify the 'sys.path' l

```
import sys
sys.path.append('/path/to/your/module')

import mymodule.
```

* Commonly used Modules :-

• math module :-
provide mathematical functions.

```
import math
print(math.sqrt(16))
print(math.pi)
print(math.sin(math.radians(90)))
```

* Random module :- provide functions to generate random numbers.

```
import random
print(random.random())
print(random.randint(1,10))
print(random.choice(['apple','banana','cherry']))
```

Emoji module:-
provides functions to work with emojis, you need to install it using 'pip install emoji'.

```
import emoji
print emoji.emojize ("python is fun: thumbs_up:")
print emoji.demojize ("python is fun fun 👍")
```

Summary

- Modules: Help organise code into manageble and reusable files.
- Create a module by saving code in a '.py' file.
- Import modules using 'import' or 'from module' import'.
- Each module has its own namespace.

* packages: Collection of related modules.

- Creates a package by organising modules in a directory with an '__init__.py' file.
- import package to use the functionalities of contained modules.

- Common modules:
- math: provides mathematical function and constant.
- Random: provides random number generation.
- Emoji provides functions to handle emojis in string.