GOVERNMENT POLYTECHNIC AMRAVATI
DIPLOMA PROGRAMME IN COMPUTER ENGINEERING

COURSE : PROGRAMMING WITH PYTHON

COURSE CODE :CM5461

Unit 4
Function

RATIONALE

- Python is used for developing desktop GUI applications, websites and web applications. Also, as a high level programming language it allows you to focus on core functionality of the application by taking care of common programming tasks. This course is designed to help the students to understand fundamental syntactic information about ‘Python’. Also it will help the students to apply the basic concepts, program structure and principles of ‘Python’ programming paradigm to build given application. The course is basically designed to create a base to develop foundation skills of programming language.

COURSE OUTCOMES (COs)

At the end of this course, student will be able to: -

- Write and execute simple ‘Python’ programs.
- Write ‘Python’ programs using arithmetic expressions and control structure.
- Develop ‘Python’ programs using List, Tuples and Dictionary.
- Develop/Use functions in Python programs for modular programming approach.
- Develop ‘Python’ programs using File Input/output operations.
- Write ‘Python’ code using Classes and Objects.

UNIT 4 :- CONTENTS

Function

- A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.
- As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called *user-defined functions*.

Function

- Defining a Function
- You can define functions to provide the required functionality. Here are simple rules to define a function in Python.
- Function blocks begin with the keyword **def** followed by the function name and parentheses (( )).

Function

- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Function

```
def functionname( parameters ):
    "function_docstring"
    function_suite
return [expression]
```

- The following function takes a string as input parameter and prints it on standard screen.

```
def printme( str ):
"This prints a passed string into this function“
 print str
return
```

Function

Calling a Function

- Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.
- Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt. Following is the example to call printme() function –

Function

```
# Function definition is here
def printme( str ):
      "This prints a passed string into this function"
     print str
 return;
# Now you can call printme function
 printme("I'm first call to user defined function!")
 printme("Again second call to the same function")
```

Function

```
I'm first call to user defined function!

Again second call to the same function
```

Anonymous functions.

- In Python, an anonymous function is a function that is defined without a name.
- While normal functions are defined using the def keyword in Python, anonymous functions are defined using the lambda keyword.
- Hence, anonymous functions are also called lambda functions.
- A lambda function in python has the following syntax.
- **Syntax of Lambda Function in python**
- lambda arguments: expression
- Lambda functions can have any number of arguments but only one expression. The expression is evaluated and returned. Lambda functions can be used wherever function objects are required.

```
# Program to show the use of lambda functions
double = lambda x: x * 2
 print(double(5))
```

- In the above program, lambda x: x * 2 is the lambda function. Here x is the argument and x * 2 is the expression that gets evaluated and returned.

- This function has no name. It returns a function object which is assigned to the identifier double. We can now call it as a normal function. The statement double = lambda x: x * 2

  is nearly the same as:

```
def double(x):
 return x * 2
```

- **Use of Lambda Function in python**

We use lambda functions when we require a nameless function for a short period of time.

- In Python, we generally use it as an argument to a higher-order function (a function that takes in other functions as arguments). Lambda functions are used along with built-in functions like filter(), map() etc.

Example use with filter()

The filter() function in Python takes in a function and a list as arguments. The function is called with all the items in the list and a new list is returned which contains items for which the function evaluates to True. Here is an example use of filter() function to filter out only even numbers from a list.

```
# Program to filter out only the even items from a list
 my_list = [1, 5, 4, 6, 8, 11, 3, 12]
new_list = list(filter(lambda x: (x%2 == 0) , my_list))
 print(new_list)
```

Example use with map()

The map() function in Python takes in a function and a list. The function is called with all the items in the list and a new list is returned which contains items returned by that function for each item. Here is an example use of map() function to double all the items in a list.

```
# Program to double each item in a list using map()
my_list = [1, 5, 4, 6, 8, 11, 3, 12]
new_list = list(map(lambda x: x * 2 , my_list))
print(new_list)
```

Output

```
[2, 10, 8, 12, 16, 22, 6, 24]
```

Python Variable:

- **Global Variables**

  In Python, a variable declared outside of the function or in global scope is known as a global variable. This means that a global variable can be accessed inside or outside of the function.

  Let's see an example of how a global variable is created in Python.

```
x = "global"
def foo():
 print("x inside:", x)
 foo()
print("x outside:", x)
x inside: global
x outside: global
```

Python Variable:

In the above code, we created x as a global variable and defined a foo() to print the global variable x. Finally, we call the foo() which will print the value of x.

What if you want to change the value of x inside a function?

```
x = "global"
def foo():
 x = x * 2
print(x)
foo()
```

The output shows an error because Python treats x as a local variable and x is also not defined inside foo().

Python Variable:

Local Variables

A variable declared inside the function's body or in the local scope is known as a local variable.

Accessing local variable outside the scope

```
def foo():
y = "local"
foo()
print(y)
```

NameError: name 'y' is not defined The output shows an error because we are trying to access a local variable y in a global scope whereas the local variable only works inside foo() or local scope.

Python Variable:

Create a Local Variable

Normally, we declare a variable inside the function to create a local variable.

```
def foo():
y = "local"
print(y)
foo()
```

Output

localLet's take a look at the earlier problem where x was a global variable and we wanted to modify x inside foo().

Python Module

- Modules refer to a file containing Python statements and definitions.
- A file containing Python code, for example: example.py, is called a module, and its module name would be example.
- We use modules to break down large programs into small manageable and organized files. Furthermore, modules provide reusability of code.

Python Module

- We can define our most used functions in a module and import it, instead of copying their definitions into different programs.
- Let us create a module. Type the following and save it as example.py
- # Python Module example
- def add(a, b):
- """This program adds two
- numbers and return the result """
- result = a + b
- return result

Python Module

- Here, we have defined a function add() inside a module named example. The function takes in two numbers and returns their sum.

```
def add( x,y):
 z=x+y
  return z
add(5,5)
```

How to import modules in Python

- We can import the definitions inside a module to another module or the interactive interpreter in Python.
- We use the import keyword to do this. To import our previously defined module example, we type the following in the Python prompt.
- import example
- example.add(4,5.5)
- 9.5

How to import modules in Python

- This does not import the names of the functions defined in example directly in the current symbol table. It only imports the module name example there.
- Using the module name we can access the function using the dot . operator. For example:
- Python has tons of standard modules. You can check out the full list of Python standard modules and their use cases. These files are in the Lib directory inside the location where you installed Python.
- Standard modules can be imported the same way as we import our user-defined modules.

How to import modules in Python

- There are various ways to import modules. They are listed below.
- **Python import statement**
- We can import a module using the import statement and access the definitions inside it using the dot operator as described above. Here is an example.
- # import statement example # to import standard module math
  import math
- print("The value of pi is", math.pi)
- When you run the program, the output will be:
- The value of pi is 3.141592653589793

How to import modules in Python

- **Import with renaming**
- We can import a module by renaming it as follows:
- # import module by renaming it import math as m
- print("The value of pi is", m.pi)
- We have renamed the math module as m. This can save us typing time in some cases.
- Note that the name math is not recognized in our scope. Hence, math.pi is invalid, and m.pi is the correct implementation.

How to import modules in Python

- **Python from...import statement**
- We can import specific names from a module without importing the module as a whole. Here is an example.
- # import only pi from math module
- from math import pi
- print("The value of pi is", pi)
- Here, we imported only the pi attribute from the math module.
- In such cases, we don't use the dot operator. We can also import multiple attributes as follows:
- >>> from math import pi, e >>> pi 3.141592653589793 >>> e 2.718281828459045

How to import modules in Python

- **Import all names**

We can import all names(definitions) from a module using the following construct:

```
# import all names from the standard module math
from math import *
print("The value of pi is", pi)
```

- Here, we have imported all the definitions from the math module. This includes all names visible in our scope except those beginning with an underscore(private definitions).
- Importing everything with the asterisk (*) symbol is not a good programming practice. This can lead to duplicate definitions for an identifier. It also hampers the readability of our code.

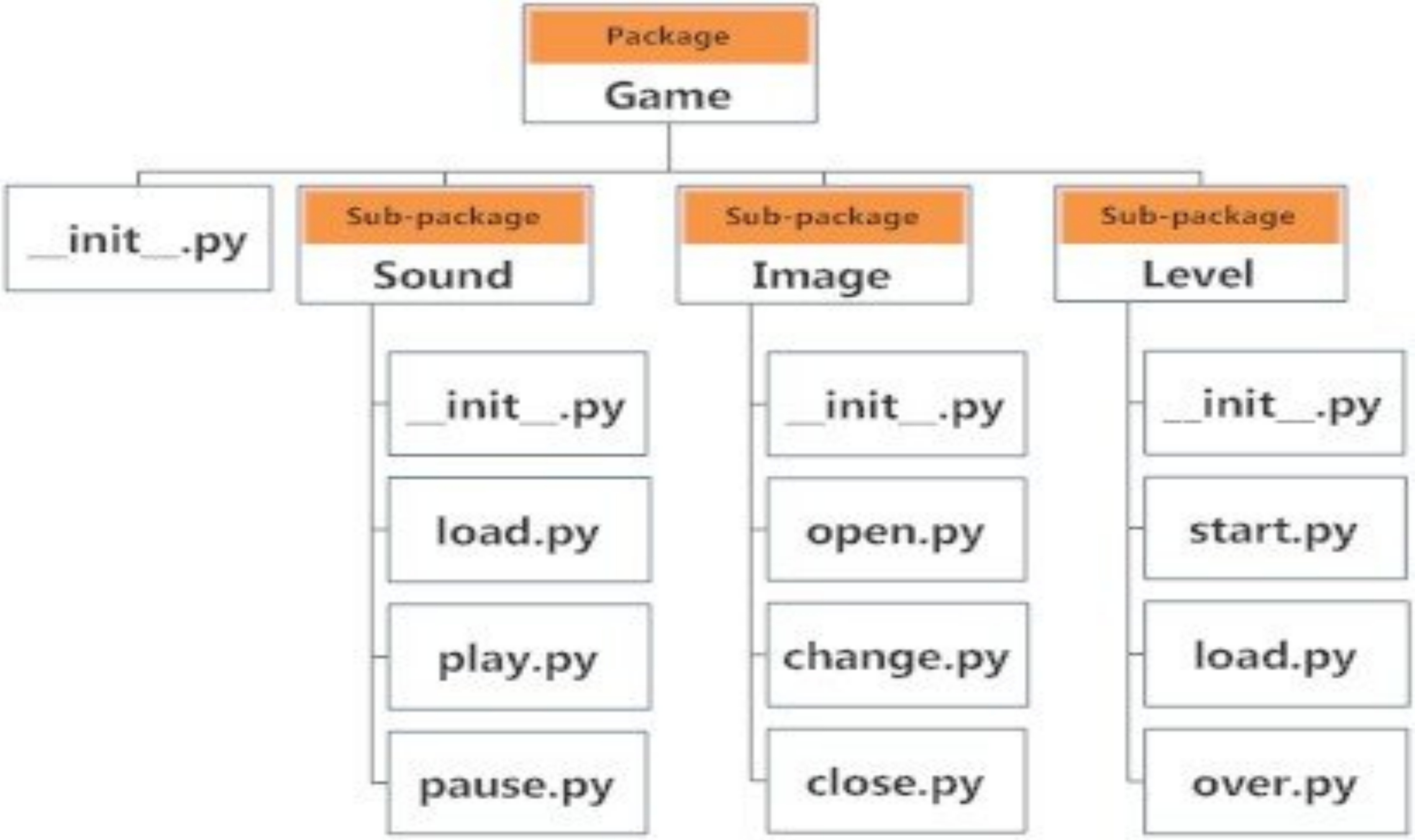
Packages in Python

- We don't usually store all of our files on our computer in the same location. We use a well-organized hierarchy of directories for easier access.
- Similar files are kept in the same directory, for example, we may keep all the songs in the "**music**" directory. Analogous to this, Python has packages for directories and modules for files.
- As our application program grows larger in size with a lot of modules, we place similar modules in one package and different modules in different packages. This makes a project (program) easy to manage and conceptually clear.

Packages in Python

- Similarly, as a directory can contain subdirectories and files, a Python package can have sub-packages and modules.
- A directory must contain a file named __init. __py in order for Python to consider it as a package. This file can be left empty but we generally place the initialization code for that package in this file.
- Here is an example. Suppose we are developing a game. One possible organization of packages and modules could be as shown in the figure below.

Packages in Python



Exceptions in Python

- Python has many built-in exceptions that are raised when your program encounters an error.(something in the program goes wrong).
- When these exceptions occur, the Python interpreter stops the current process and passes it to the calling process until it is handled. If not handled, the program will crash.

Exceptions in Python

- For example, let us consider a program where we have a function A that calls function B, which in turn calls function C. If an exception occurs in function C but is not handled in C, the exception passes to B and then to A.
- If never handled, an error message is displayed and our program comes to a sudden unexpected halt.

Catching Exceptions in Python

- In Python, exceptions can be handled using a try statement.
- The critical operation which can raise an exception is placed inside the try clause. The code that handles the exceptions is written in the except clause.
- We can thus choose what operations to perform once we have caught the exception. Here is a simple example.

Catching Exceptions in Python

```
# import module sys to get the type of exception
import sys
randomList = ['a', 0, 2]
for entry in randomList:
    try:
print("The entry is", entry)
 r = 1/int(entry)
break
     except:
```

Catching Exceptions in Python

- In this program, we loop through the values of the randomList list. As previously mentioned, the portion that can cause an exception is placed inside the try block.
- If no exception occurs, the except block is skipped and normal flow continues(for last value). But if any exception occurs, it is caught by the except block (first and second values).
- Here, we print the name of the exception using the exc_info() function inside sys module. We can see that a causes ValueError and 0 causes ZeroDivisionError.

Catching Exceptions in Python

- Since every exception in Python inherits from the base Exception class, we can also perform the above task in the following way

Catching Exceptions in Python

- # import module sys to get the type of exception
- import sys
- randomList = ['a', 0, 2]
- for entry in randomList:
- try: print("The entry is", entry) r = 1/int(entry)
- break

except

Exception as e:

print("Oops!", e.__class__, "occurred.")

Catching Exceptions in Python

- In the above example, we did not mention any specific exception in the except clause.
- This is not a good programming practice as it will catch all exceptions and handle every case in the same way. We can specify which exceptions an except clause should catch.
- A try clause can have any number of except clauses to handle different exceptions, however, only one will be executed in case an exception occurs.
- We can use a tuple of values to specify multiple exceptions in an except clause. Here is an example pseudo code.

Catching Exceptions in Python

```
try:

# do something

Pass

 except ValueError:

 # handle ValueError exception pass

Except (TypeError, ZeroDivisionError):

# handle multiple exceptions

 # TypeError and ZeroDivisionError

pass

except:
```