Object Oriented Programming

BE(CSE) II-Semester

Prepared by
S. Durga Devi,
Assistant Professor,
CSE,CBIT

Unit-II

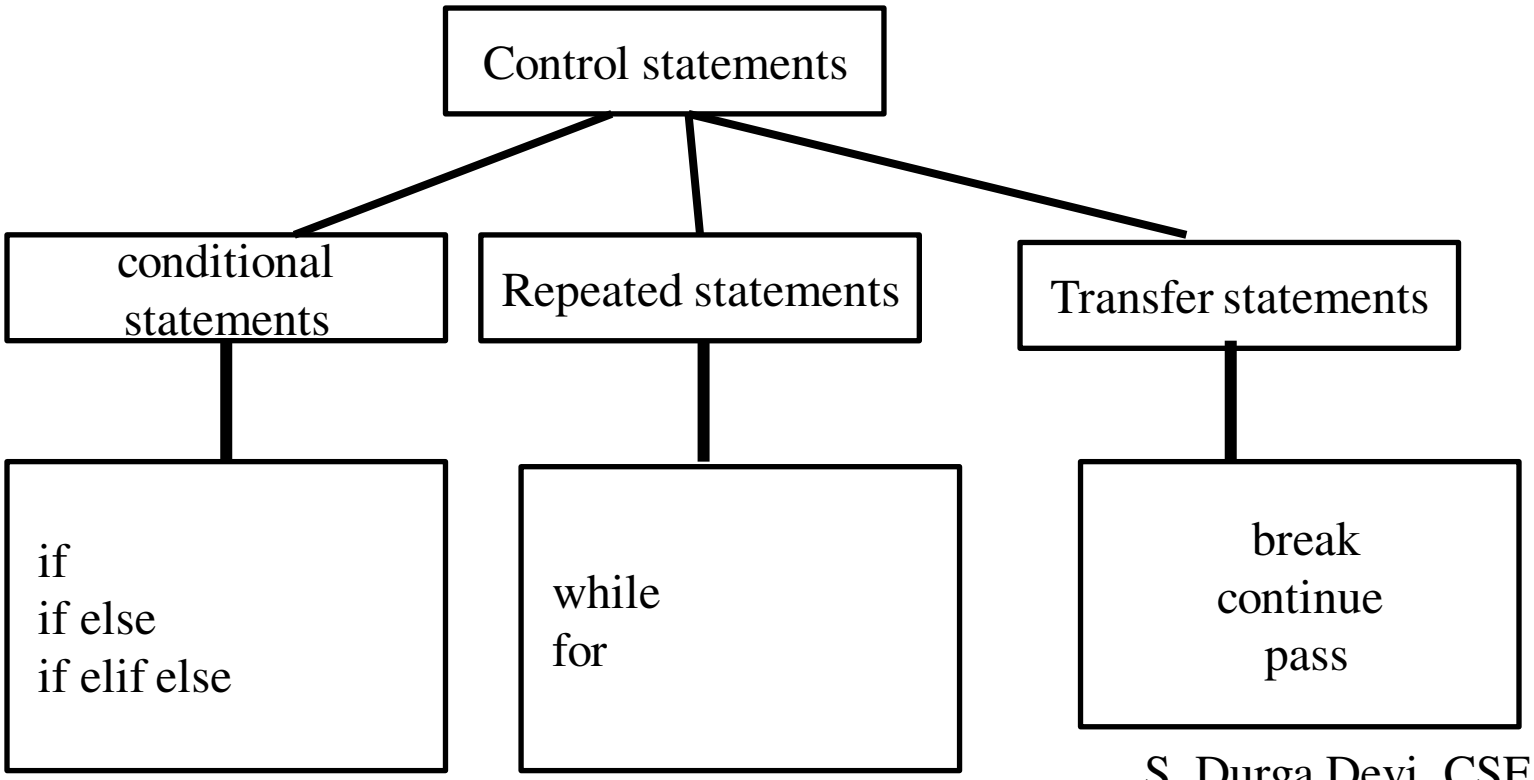
- **Decision Control Statement:**
  - Selection/Conditional Branching,
  - Loop Control Structures
  - Nested loops.
- **Functions and Modules:**
  - Uses of functions
  - Function definition
  - function call
  - Variable scope and Lifetime
  - Recursion
  - Lambda functions
  - Recursive Functions
  - Modules
  - Packages.

Decision Control Statement:

Control flow statements: determines order in which statements to be executed

Three types of decision control statements

1. Conditional statements
2. Repeated statements
3. Transfer statements



control statements

1. Conditional/ selection statements

if

if - else

if – elif – else

- No curly braces in python
- Colon is required after the condition in if and elif statements

```
if condition:
        statement1
elif condtion:
          statement2
else:
        statement3
    ---
```

Example on conditional statement

```
# largest of three numbers
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
num3 = float(input("Enter third number: "))

if (num1 >= num2) and (num1 >= num3):
   largest = num1
elif (num2 >= num1) and (num2 >= num3):
   largest = num2
else:
   largest = num3

print("The largest number between",num1,",",num2,"and",num3,"is",largest)
```

Repeated statements

- In python we have only for and while loop
- There is no do- while loop in the python
- **for loop**

```
s="CBIT HYDERABAD"
count=0
for x in s:
    print(x)
    count=count+1
print("The no of characters", count)
```

Example-2

```
l=[10,20,30,40]
for x in l:
        print(x)
```

Example3

```
s="DURGADEVI"
 i=0
 for x in s:
     print("the character at ",i,"is",x)
     i=i+1    # there is no i++ in python
```

print 20 to 1 in descending order

```
 for x in range(20,0,-1)
            print (x)
```

print odd numbers from 0 to 20

```
  for x in range(20):
          if x%2!=0:
              print(x)
```

or

```
   for x in range(1,20,2)
            print(x)
```

Note: range(20) value is in between 0 to 19

while loop

- If you know number of iterations in advance then use for loop
- If you don't know number of iterations then use while loop
- Suppose put a sugar in the tea cup and stir it until sugar dissolve in the tea. We don't know how many no of times we need to stir it once and check whether sugar dissolved or not
- Example

```
# print numbers from 1 to 10
        x=1
          while x<=10:
                  print(x)
                  x=x+1
```

Nested loops

```
# print the below pattern using nested loop
*
* *
n= int(input("enter n value"))
 for  i in range(1,n+1):
            for  j in range(1,i+1):
                        print("*", end =' ')
             print("\n")
```

Write a program to display the right triangle star

```
n = int(input("Enter number of rows:"))
for i in range(1,n+1):
                    for j in range(1,i+1):
                                        print("*",end=" ")
                    print("\n")
print()
```

```
Enter number of rows:5
*

* *

* * *

* * * *

* * * * *
```

Transfer statements

1. break 2. continue 3. pass

1. break

exit from the nearest loop. When break statement encounter control passed to the statement follows the loop

Ex for i in range(10):

```
    if(i== 7):
        print("stop")
        break;
    print(i)
```

else clause

- else clause can also be associated with the for loop
- else is encounter when for loop is terminated normally
- If for loop is terminated by break else will not executed.
- Example

```
# program to find the given number is prime or not
num=eval(input("enter a number"))
for i in range(2,num):
                if(num%i==0):
                                print("not a prime")
                                break
else:
                print("prime number")

enter a number20
not a prime
```

- # print prime numbers from 1 to 100 using else clause

```
for num in range(1,101):
                for i in range(2,num):
                                if(num%i==0):
                                                break
                else:
                                print(num,end=' ')
```

```
1 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
>>>
```

```
cart=[60,240,100,75,200,500]
total=0
for item in cart:
     total=total+item
     if total>1000:
          print("free shipping charges")
          break
     print("added item is",item)
else:
     print("shipping charges 100 extra total is ",total+100)
```

continue statement

skips the current iteration and continue the next iteration

Example

```
    # print add numbers
for x in range(10):
        if(x%2==0)
            continue
        print(x)
```

pass statement

- pass is a keyword in python
- If a block in a program does not do anything that block is defined with pass keyword.

Example

```
if true:
        else:
                print("hello")
```

- above statements will give error. As you have not defined anything for the if block.
- To avoid such errors if block is defined with pass keyword

```
if true: pass
        else:
                print("hello")
```

pass statement

1. It is an empty statement like null statement
2. It does not do anything.

```
def f1():
    print("hello")
def f2(): pass
f1()
f2()
```

Functions

- Function is a group of statements used to perform a specific and well defined task
- These statement are defined as a single unit and we can call any no of times based on our requirement without rewriting. This single unit is called function

Two types of functions:

1. Built in function: provided by python
2. User defined functions: user can define his own functions as per his requirements are called user defined functions

- Every function encapsulates a set of operations and when called it returns the information to the calling program or function.

functions

Syntax

```
def  functionname(parameters):
          -----
            -----
          return
```

```
def  hello:
     print("hello world");

 hello();
```

1.Keyword def marks the start of function header.
2. A function name to uniquely identify it.
3.A colon (:) to mark the end of function header.
4. Optional documentation string (docstring) to describe what the function does.
5. One or more valid python statements that make up the function body.
6.Statements must have same indentation level (usually 4 spaces).
7. Return statement is optional to return a value from the function.

- increases reusability
- in other languages function is also called a method, procedure or sub routines

return statement

- return statement: function can take input values as parameters and executes business logic and returns output to the caller with return statement.
- If no return value in a function then default return value is None.

```
def f1():
                print("hai")
f1()
print(f1()) #None
```

- return statement can return multiple values simultaneously in python

```
def sum_sub(a,b):
                sum=a+b
                sub=a-b
                return sum,sub
x,y=sum_sub(40,30)
print("sum=",x)#sum=70
print("sub=",y)#sub=10
```

Types of arguments

```
def f1(a,b):
        .......
        .......
f1(10,20)
```

- a,b are formal arguments and 10, 20 are actual arguments
- There are 4 types of actual arguments in python

1. Required arguments/ positional arguments
2. Keyword arguments
3. Default arguments
4. Variable length arguments

1. Required or positional arguments

- The arguments are passed to function in correct positional order.
- No of arguments should be same as no of arguments specified in function definition.

Example:

```
def f1(a,b):
        print(a-b)
f1(100,20)
```

Note: if you change the order of actual arguments result will be changed.

2. Keyword arguments

- In python we can pass argument values by keyword i.e by parameter name.
- Values assigned to formal arguments according to their name rather than position
- Order not important but no of arguments must be matched.

```
def employee(name,id,salary):
                print(name,id,salary)
employee(salary='2L',name="Durga Devi",id=10556)
```

- We can use both positional and keyword arguments simultaneously but first we have to take positional arguments and then keyword arguments otherwise gives error.

- **def wish(name,msg):**
- **print("Hello",name,msg)**
- **wish("Durga Devi","GoodMorning") ==>valid**
- **wish("Durga Devi ",msg="GoodMorning") ==>valid**
- **wish(name="Durga Devi","GoodMorning") ==>invalid**
- **SyntaxError: positional argument follows keyword argument**

3. Default arguments

- We can provide default values for positional arguments.
- If you are not passing any value while calling a function then default value will be considered.

Example

```
def wish(name="Guest"):
        print("Hello",name,"Good Morning")

 wish(“CBIT")
wish()
```

Output:

Hello CBIT Good Morning
Hello Guest Good Morning

Default arguments

- After default arguments we should not take non default arguments
- def wish(name="Guest",msg="Good Morning"): ===>Valid
- def wish(name,msg="Good Morning"): ===>Valid
- def wish(name="Guest",msg): ==>Invalid
- SyntaxError: non-default argument follows default argument

4. Variable length arguments

- We can pass any number arguments to our function such type of arguments are called variable length arguments.
- Variable length arguments represent with *
- We can pass any number of arguments including zero number. Internally all these values represented as tuple.

```
def sum(*n):
                    total=0
                    for x in n:
                                        total=total+x
                    print("the sum is=",total)
sum()#0
sum(10,20)#30
sum(10,20,30)#60
```

4. Variable length arguments

- We can mix variable length arguments with positional arguments

```
Def fun(n,*s):
        print(n)
        for x in s:
                print(x)
fun(10) #10
fun(10,20,30,40) # n=10, s=(20,30,40)
fun(10,'A',30,'B')  #n=10, s=('A',30,'B')
```

- **After variable length argument, if we are taking any other arguments then we should provide values as keyword arguments.**

```
def f1(*s,n1):
        for s1 in s:
                print(s1)
        print(n1)

f1("A","B",n1=10)
```

Output

A
B
10

4. Variable length arguments

- Note: We can declare key word variable length arguments also.
- For this we have to use **.

- Example

def f1(**n):

- We can call this function by passing any number of keyword arguments. Internally these keyword arguments will be stored inside a dictionary.
- Exmaple

```
def sample(**keywordargs):
    for k,v in keywordargs.items():
        print(k,":",v,end=" ")
    print("\n")
sample(a=10,b=20,c=30,d=40)
sample(name="Durga Devi",rollno=505,subject="python",marks=99)
```

```
a : 10 b : 20 c : 30 d : 40

name : Durga Devi rollno : 505 subject : python marks : 99
```

Scope and lifetime of variables

- Scope: determines which parts of the program variable is accessed.
- life time of variable: how long variable exist.

- Two types of variables:

1. **Global variable:** variables declared out side of the function is called global variables.
2. **Local variables:** declared inside of a function:
   scope and life of time of local variables are within the function.

1. Global variables

- Variables which are declared outside of a function are called global variables.
- Global variables can be accessed by the all the functions of a module.

```
a=10
def f1():
    print(a)
def f2():
    print(a)
f1()
f2()

output
10
20
```

2. Local variables

- Variables defined inside a function is called local variables
- Local variables can be accessed within the function as scope and life time of local variables is within the block
- Can we use local and global variables names same?
- Note: if local and global variables has name we can access global variables inside of a function using **globals()**
- **Ex**

```
a=20
def f1():
    a=100
    print(a)#100
    print(globals()['a'])#20
f1()
```

global keyword

- global keyword is used for the following purposes

1. To declare global variable inside a function
2. To update global variable value by all the functions

Ex1

```
a=10
def f1():
    a=234
    print(a)
def f2():
    print(a)
f1()
f2()
```

Output

```
234
10
```

Ex2

```
a=10
def f1():
    global a
    a=234
    print(a)
def f2():
    print(a)
f1()
f2()
```

output

```
234
234
```

Ex3

```
def f1():
    global a
    a=234
    print(a)
def f2():
    print(a)
f1()
f2()
output
234
234
```

Recursion

- Recursion: A function which calls itself is called recursion.
- Recursion solves smaller tasks by calling itself.
- Recursive function uses two case

1. Base case: where recursion should be terminated
2. Recursive case: function calls itself to perform sub tasks.

Why recursion need?

- There are some problems which are difficult to solved through iterative methods can be solved by recursion. Such as merge sort, binary search method, tree traversals etc.
- Recursion reduces the code and improves readability.

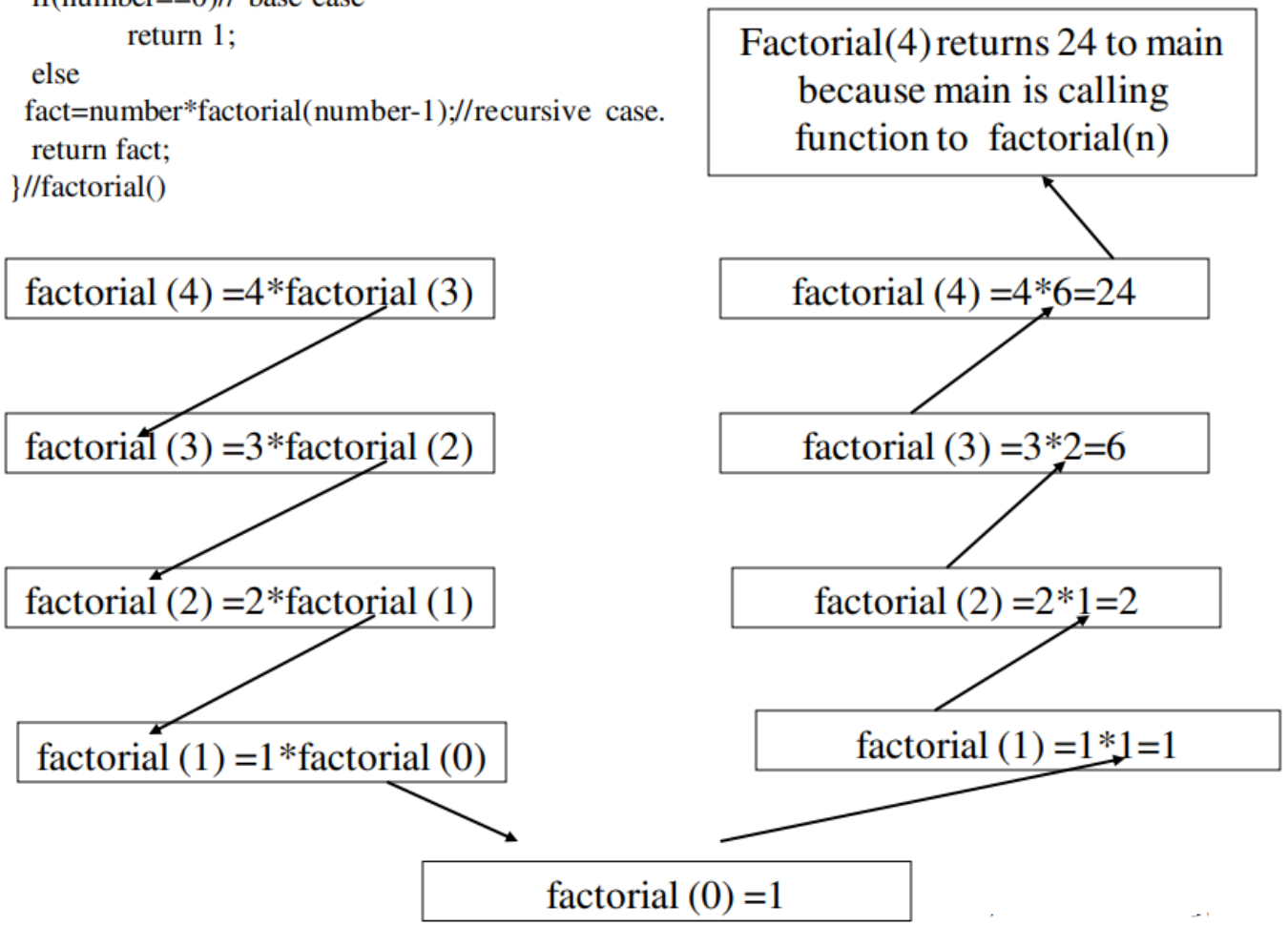
Example

- To find factorial of a given number recursive function is

  n!=1, if n=0 //base case

  n!=n*(n-1)!, if n>0 // recursive case.

```
factorial(int number)
 {                                  number=4
  int fact;
   if(number==0)// base case
            return 1;
   else
   fact=number*factorial(number-1);//recursive case.
   return fact;
 }//factorial()
```



Differences between recursive and iterative methods.

Recursive method	Iterative method
Reduces the code	Length of the code is more
Speed of recursive methods are slow	Faster
terminated when it meets base condition	terminated when the condition is false
Each recursive call requires extra space to execute. Ex stack	does not require any extra space
If recursion goes into infinite , the program run out of memory and results in stack overflow	Goes to infinite loop
solution to some problems are easier with recursion	solution to problem may not always easy

Factorial of a number using recursive method

factorial.py - D:/MyPPTs/LPSL-2018/UNIT-V(Python)/PythonPrograms/factorial.py (3.6.4)

File Edit Format Run Options Window Help

```
#factorial of a number using python
def factorial(x):
                if x==1:
                                return 1;
                else:
                                return (x*factorial(x-1));
x=4
print("factorial of a given number is =", factorial(x))
```

Lambda function/ anonymous function

- A function without name is called anonymous function or lambda function.
- Anonymous functions are defined with lambda keyword.
- Instant use .

syntax

```
lambda arguments: expression
```

- lambda keyword is followed by any number of arguments but have one expression.
- expression is evaluated and returns the result.
- Lambda functions are used when you want to get function object.

Example

```
# double a  number using lambda function
x=lambda x:x*2;
print(x(10));
# output   20
```

Lambda functions

- Lambda Function internally returns expression value and we are not required to write return statement explicitly.
- Sometimes we can pass function as argument to another function. In such cases lambda functions are best choice.
- We can use lambda functions very commonly with **filter(),map() and reduce()** functions , because these functions expect function as argument.

1. filter() : filter function is used to filter required values from the given sequence based on some condition.

- Syntax

filter(functionname, sequence)

- For every element in sequence specified function will be applied
- Sequence could be list, tuple, set and dict
- Ex
- Write a program to filter even nos from the list using filter function.

```
# without lambda function
def even(x):
                if x%2==0:
                                return True
                else:
                                return False

l=[1,2,3,4,5,6,7,8,9,20,21,31]
l1=list(filter(even,l))
print(l1)
```

```
[2, 4, 6, 8, 20]
```

```
# with lambda function
l=[1,2,3,4,5,6,7,8,9,20,21,31]
l1=list(filter(lambda x:x%2==0,l))
print(l1)
```

```
[2, 4, 6, 8, 20]
```

2. map() function: for each element in sequence , apply some functionality and generate new list.

- Input list and output list contains same no of elements.
- Example

```
# double values in list using map function
l=[2,3,4,5]
l1=list(map(lambda x: x*x,l))
print(l1)
```

```
[4, 9, 16, 25]
```

write a program to multiply two given list values using map function

```
# double values in list using map function
l1=[2,3,4,5]
l2=[3,2,5,4]
l3=list(map(lambda x,y: x*y,l1,l2))
print(l3)
```

```
[6, 6, 20, 20]
```

3. reduce() function

- reduce() function reduces sequence of elements into single element using some function
- reduce() function gives only single value
- syntax
- reduce(functionname,sequence)
- reduce () function available in funtools module

```
import functools
num=[1,2,3,4]
sum=functools.reduce(lambda x,y:x+y,num)
print(sum)
```

Output
10

Function aliasing

- We can use another name to a function

```
def employee_details(empid,empname):
                    print(empid,empname)
e=employee_details
e(123,"CBIT")
employee_details(10556,"Durga Devi")
print(id(e))
print(id(employee_details))
```

```
123 CBIT
10556 Durga Devi
47349240
47349240
```

Function aliasing

- If we delete one name still we can access that function using alias name

```
def employee_details(empid,empname):
                    print(empid,empname)
e=employee_details
e(123,"CBIT")
del employee_details
employee_details(10556,"Durga Devi")#NameError
print(id(e))
print(id(employee_details))
```

Nested functions

- A function defined inside of a another function.

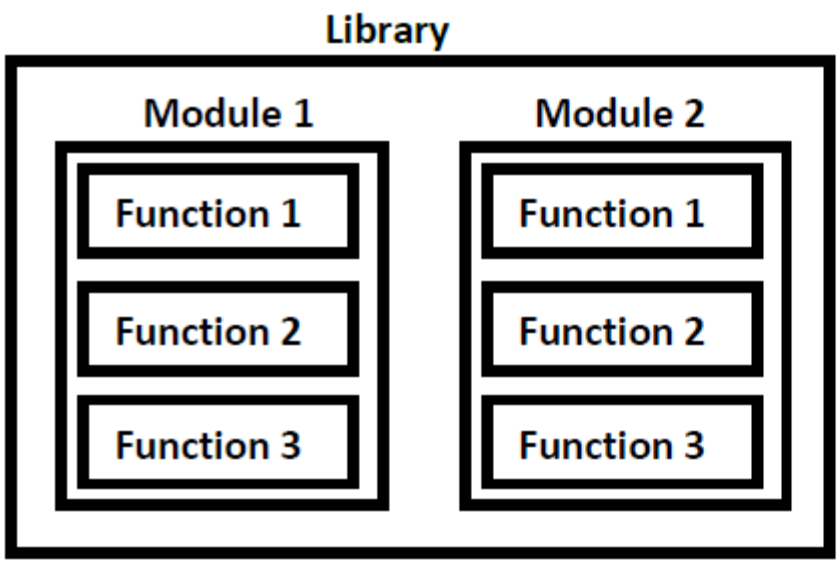
```
def outer():
    print("we are in outer function")
    def inner():
        print("we are in inner functions")
    inner()
outer()
```

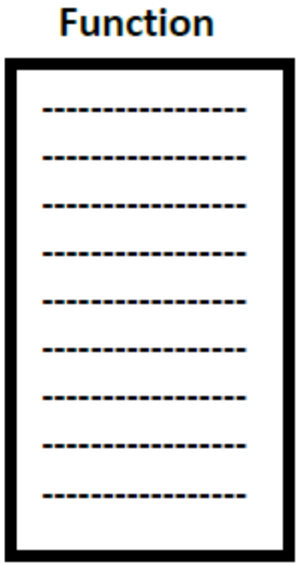
Inner() # gives NameError

Note:since inner function is local to outer(),
inner function cant be called out side of the function

modules

- function: a group of instructions under a single name used to perform a specific task
- Module: a group of functions saved under a single file is called module
- Libraries : a group of modules are called libraries.





modules

- A module consists of group of functions, classes , variables and constants.
- Every python file acts as a module(.py file)
- Example

```
#test.py
s="CBIT"
def Hello():
                    print("hello")
def wish():
                    print("good morning")
```

- If you want to use members of a module in our program we must import that module
- general meaning of import get from some where

Syntax

```
import module_name
```

- Import statement should be first line of your program
- Members of a module can accessed by using module name followed by membername

```
modulename.variablename
modulename.funtionname()
```

Example

```
import test
print(test.s)
test.Hello()
test.wish()
```

from ….. import

- we can import particular members of module by using from… import .
- Advantage is we can access members directly without using module name
- Example

```
from test import *
print(s)
Hello()
```

How many ways we can import module

1. import modulename
2. import modulename as m # rename module
3. import module1,module2…. # import more than module
4. import module1 as m1,module2 as m2 #rename modules
5. from modulename import membername# imports specific member from the module
6. from module import member1,member2, member3
7. from modulename import * # import all members
8. from module import member1 as m1, member2 as m2 # we alias membernames also.

dir() funtion

- dir() is a builtin function is used to list out all members in a current module or specified module
- Syntax
- dir(modulename) # list out members of specified module
- dir() # list out members of current module.

dir() funtion

```
a=100
b=200
def add(a,b):
      print(a+b)
def sub(a,b):
      print(a-b)
def product(a,b):
      print(a*b)
print(dir())
```

```
['__annotations__', '__builtins__', '__doc__', '__file__', '__loader__', '__name
__', '__package__', '__spec__', 'a', 'add', 'b', 'product', 'sub']
>>>
```

Note: when you execute a python program(module) python interpreter uses default members or properties internally.

Special variable __name__

- For every Python program , a special variable __name__ will be added internally.
- This variable stores information regarding whether the program is executed as an individual program or as a module.
- If the program executed as an individual program then the value of this variable is __main__
- If the program executed as a module from some other program then the value of this variable is the name of module where it is defined.
- Hence by using this __name__ variable we can identify whether the program executed directly or as a module.

Example on __name__ variable

Execute below program directly

```
#module1.py
def f1():# directly executed
                if __name__=='__main__':
                                print("the code executed directly as a program")
                                print("the value of __name__:",__name__)
                else:
                                print("the code executed directly as a module")
                                print("the value of __name__:",__name__)
f1()
```

```
the code executed directly as a program
the value of __name__: __main__
>>>
```

Test.py

```
import module1
print("from test we are executing module1 f1()")
module1.f1()
```

```
the code executed directly as a module
the value of __name__: test
from test we are executing module1 f1()
the code executed directly as a module
the value of __name__: test
>>>
```

Reload module

- By default module will be loaded only once even though we are importing multiple times.
- Example
- Test.py

```
import module1
import module1
import module1
```

Note: In the above program, every time updated version of module1 will be available to our program.
-If you made changes in the modules after importing in a program, changes not reflected in the imported program.
- use reload() function to get the updated version of the module.

Example on reload()

module1.py

```
def wish(name):
        print("Happy Birthday Dear",name)
```

```
>>> import module1
>>> module1.wish("Devi")
Happy BirthDay Devi
```

module1.py

```
def wish(fname,lname):
        print("Happy Birthday Dear",fname,lname)
```

```
>>> module1.wish("Durga","Devi")
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    module1.wish("Durga","Devi")
TypeError: wish() takes 1 positional argument but 2 were given
>>> import imp

Warning (from warnings module):
  File "<pyshell#3>", line 1
DeprecationWarning: the imp module is deprecated in favour of importlib; see the
 module's documentation for alternative uses
>>> imp.reload(module1)
<module 'module1' from 'C:/Users/DURGA DEVI/Desktop/CSE\\module1.py'>
>>> module1.wish("Durga","Devi")
Happy BirthDay Durga Devi
>>>
```

math module

math module provides several mathematical operations

- sqrt()
- ceil()
- floor()
- sin()
- tan()
- cos()
- pow()

- To know more about math module use following commands

import math

help(math)

random *module*

- random module contains several functions to generate random numbers.
- random module used to develop game applications and generating OTP for authentication.
- To find list of methods

```
import random
help(random)
```

1. random() : generates float value from 0 to 1(not inclusive) 0<x<1

```
from random import *
for  i in range(10):
        print(random)
```

```
0.502336054801914 2
0.262106818767375 6
0.693280398153125 7
0.153726529120835 06
0.209805490136984 8
0.815173847404097 3
0.723135240869846 8
0.377343475120914 93
0.732562897523939 7
0.830132156447475 8
```

2. randint(): generates random integer numbers

Example

```
from random import *
for i in range(10):
        print(randint(1,50))
```

3. **uniform():** random float values between two given numbers.(not inclusive)

Example

```
  from random import *
for i in range(10):
        print(uniform(1,50))
```

-> random() : float values in between 0 and 1 (not inclusive)

-> randint(): int values in between two values (inclusive)

-> uniform(): float values in between two given values (not inclusive)

4. randrange([start],end,[step]): generates integers from start to end-1 and step specifies no elements to skip.

- Start argument is optional default is 0
- Step argument is optional default is 1
- randrange(10): generates 0 to 9
- randrange(1,10): generates 1 to 9
- randrange(1,10,2): generate numbers 1,3,5,7,9

Example

```
from random import *
    for i in range(10):
        print(randrange(10))
```

5. choice(sequence of elements): returns random elements from the given list or tuple

```
from random import *
list1=['sunny','munny','bunny','tanny']
for i in range(10):
        print(choice(list1))
```

List Comprehensions

Write a program to find a square of first 5numbers in a list

```
list1=[]
for i in range(5):
      x=int(input())
      x=x*x
      list1.append(x)
print(list1)
```

- Syntax:

list1=[expression for x in sequence]

Example find a square of first 10 numbers in a list.

L1=[x*x for x in range(1,11)]

print(L1)

Name space in python

- Name space:
- Names are used to identify an object such as variable name, function name, class name etc.
- Name space is system which controls all the name we used in our program .
- It ensures that the names are unique to avoid name conflicts.
- Name space useful when we are developing large programs to avoid name conflicts.

Types of name space

1. Built in name space: contains built in function names, attributes, constants etc.
2. Local name space: contains identifiers in a currently executing function
3. Global name space: contains identifiers in a currently executing modules.

- Example

```
# module1
    def display():
            print("hai")
# module2
    def display():

print("hello")
```

```
#test.py
import module1
import module2
 display()# ambiuous reference for
                identifier display
# to avoid this
 module1.display()
 module2.display()
```

Module private variables

- In python, all identifiers are public by default. Means all the identifiers are accessed by other module that imports it.
- if you want to restrict the access means do not access identifiers out side of the module declare those identifiers as private
- Private identifiers are defined with two __ (double underscore symbols).

- #module1.py

```
__x=20
__y=30
 def fun1():
            print(x,y)
#test.py
 from module1 import *
 print(x,y)
 fun1()
# generates an error
```

Packages

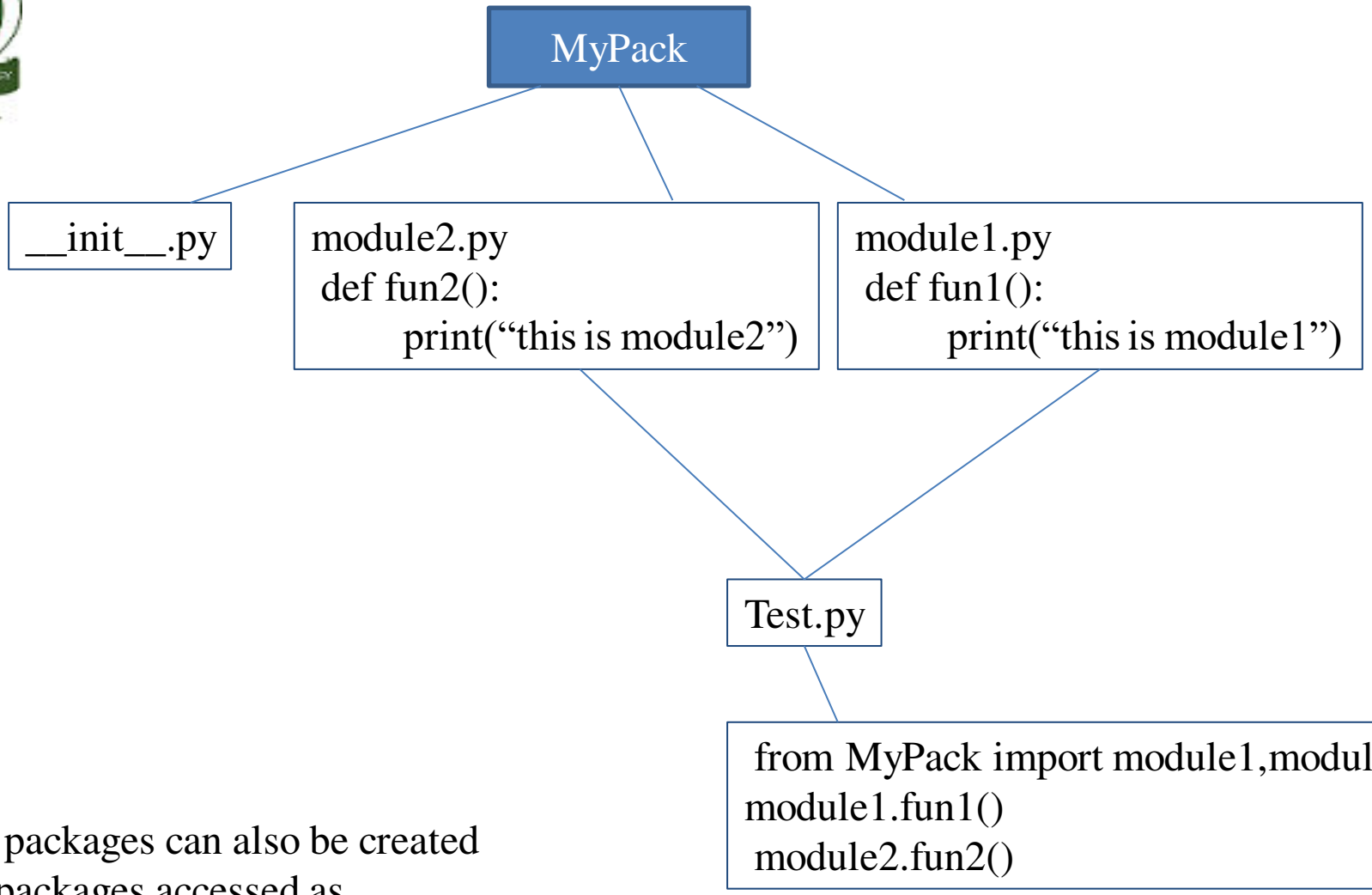
- Package is a collection of modules and sub packages.
- Package is a folder or directory.
- Each package in python is a directory which must contains a special empty file is called __init__.py.
- __init__.py indicates that this folder is a python package.

Packages

- **How to create packages?**
- 1. create a folder with some name(MyPack)
- 2. MyPack-> __init__.py(must be empty)
- 3. MyPack-> module1.py-> fun1()
- 4. MyPack->module2.py-> fun2()
- 5. create a python file Test.py outside of a MyPack

- Advantage of packages:

1. Resolves naming conflicts(all identifiers are unique)
2. Improves the modularity(grouping up the related modules into single unit)
3. Provides the security



- sub packages can also be created
- sub packages accessed as
import packagename.subpackagename

Command line arguments

- Passing arguments to a program during run time from the command prompt is called command line arguments.
- The Python **sys** module provides access to any command-line arguments via the **sys.argv**.
- This serves two purposes −

1. sys.argv is the list of command-line arguments.
2. len(sys.argv) is the number of command-line arguments.

- Type of argv variable is list type.

Example on command line arguments

commandEx.py

```
#add two number using command line arguments
import sys
a=int(sys.argv[1])
b=int(sys.argv[2])
print("sum=",a+b)
```

How to execute command line arguments

- Open command prompt
- give below command
- py commandEx.py 10 20

```
C:\Users\hp\Desktop>py CommandEx.py 10 20
sum= 30

C:\Users\hp\Desktop>
```

Questions

- What is purpose of else clause with loop? Explain with one example.
- What is the list compression ?what is the use of it?
- Can we use positional arguments and keyword arguments simultaneously? If yes justify your answer.
- What are the types of the arguments can be passed to a function?
- Extract the prime numbers from the given list of numbers using filter() method and lamda functions .
- Write a code for generating square of list of given numbers using list comprehension.
- What are the advantage of packages?
- Explain command line arguments using an example?
- Differentiate between iterative method and recursive method.
- Write a code to find fibonacci series using recursion?
- What is purpose of pass keyword?
- What is anonymous function? When anonymous function is used?
- Create your own module to perform Binary search method .
- How many ways modules can be imported in current program?
- To get the updated version of the module in current program what method is used and what is the purpose of that module?
- Case Study: create a package CBIT , in that create a module named as branches.py , define a function list_of_branches() and print what are the branches available in CBIT. In CBIT package create a sub package CSE, create a module named as Cse_specializations.py, define a function specializations() and print list of specializations provided by CSE Dept. Create a file college.py to access the modules branches.py and cse_specializations.


CHAITANYA BHARATHI
INSTITUTE OF TECHNOLOGY
1979