Academic Year:2023-24 SAP ID:

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJ19ITL406 **DATE: 15-02-2024**

COURSE NAME: Programing Laboratory 2 (Python) **CLASS:** S.Y. B.Tech

NAME: ANISH SHARMA SAP ID: 60003220045

EXPERIMENT NO. 3

CO/LO: CO1, CO2.

AIM / OBJECTIVE:

a. Write python programs to demonstrate applications of different decision-making statements and loops.
b. Write a Python program to implement Functions and Recursion, Function decorators.

DESCRIPTION OF EXPERIMENT:

If-Else statements in Python are part of conditional statements, which decide the control of code.

1. The if statement

Syntax:

```
if condition:
   # Statements to execute if
   # condition is true
```

```
if condition:
   statement1
statement2
# Here if the condition is true, if block
# will consider only statement1 to be inside
# its block.
```

2. The if-else statement

Syntax:

```
if (condition):
    # Executes this block if
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

```
    # condition is true
else:
    # Executes this block if
    # condition is false
```

3. The nested-if statement

Syntax

```
if (condition1):
   # Executes when condition1 is true
   if (condition2):
      # Executes when condition2 is true
   # if Block is end here
# if Block is end here
```

4. The if-elif-else ladder

Syntax

```
if (condition):
    statement
elif (condition):
    statement
.
.
else:
    statement
```

Example:

```
i = 20
if (i < 15):
    print("i is smaller than 15")
    print("i'm in if Block")
else:
    print("i is greater than 15")
    print("i'm in else Block")
print("i'm not in if and not in else Block")
```

Output:

```
i is greater than 15
i'm in else Block
i'm not in if and not in else Block
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

For Loops in Python

Python For loop is used for sequential traversal i.e. it is used for iterating over an iterable like String, Tuple, List, Set.

For Loops Syntax

```
for var in iterable:

   # statements
```

Example:1

```
l = ["geeks", "for", "geeks"]
for i in l:
    print(i)
```

Output :

```
geeks

for

Geeks
```

Example:2

```
d = dict()

d['xyz'] = 123
d['abc'] = 345
for i in d:
    print("% s % d" % (i, d[i]))
```

Output:

```
Dictionary Iteration
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

```
xyz  123

abc  345
```

Example: 3

```
s = "Geeks"
for i in s:
    print(i)
```

Output:

```
String Iteration

G

e

e

k

s
```

Example

```
for i in range(0, 10, 2):
     print(i)
```

Output :

```
0

2

4

6

8
```

Example

```
for i in range(1, 4):
     for j in range(1, 4):
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

```
print(i, j)
```

Output :

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

Example

```
fruits = ["apple", "banana", "cherry"]
colors = ["red", "yellow", "green"]
for fruit, color in zip(fruits, colors):
    print(fruit, "is", color)
```

Output :

```
apple is red
banana is yellow
cherry is green
```

Example

```
t = ((1, 2), (3, 4), (5, 6))
for a, b in t:
    print(a, b)
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

Output :

```
1 2

3 4

5 6
```

Example

```
# Prints all letters except 'e' and 's'
for letter in 'geeksforgeeks':


     if letter == 'e' or letter == 's':
          continue
     print('Current Letter :', letter)
```

Output:

```
Current Letter : g
Current Letter : k
Current Letter : f
Current Letter : o
Current Letter : r
Current Letter : g
Current Letter : k
```

Python While Loop is used to execute a block of statements repeatedly until a given condition is satisfied. When the condition becomes false, the line immediately after the loop in the program is executed.

Syntax of while loop in Python

```
while expression:
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

```
statement(s)
```

Example:

```
# Python program to illustrate

# while loop

count = 0

while (count < 3):

     count = count + 1

     print("Hello Geek")
```

Output

```
Hello Geek

Hello Geek

Hello Geek
```

Example:

```
age = 28

# the test condition is always True

while age > 19:

     print('Infinite Loop')
```

B. Python Functions is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

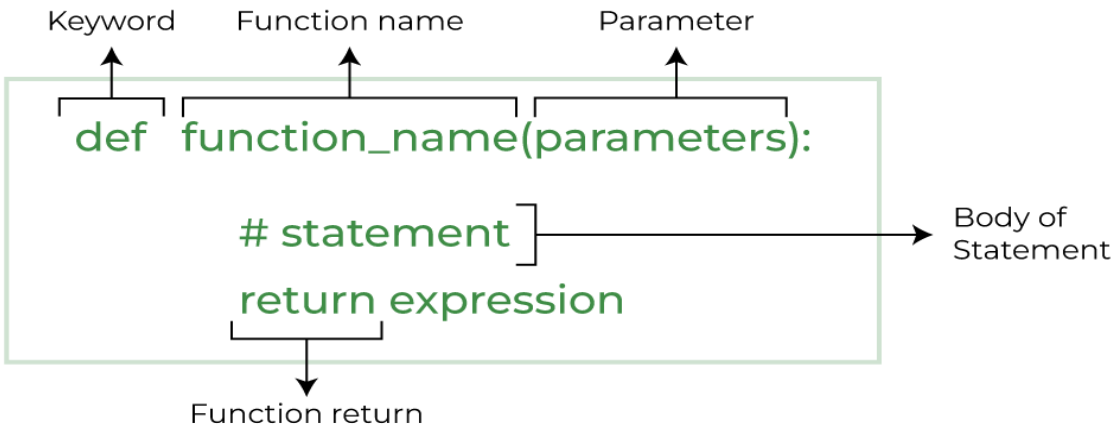
Some **Benefits of Using Functions**

- Increase Code Readability
- Increase Code Reusability

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

Python Function Declaration

The syntax to declare a function is:



Example:

```
# A simple Python function
def fun():
     print("Welcome to Python")


# Driver code to call a function
fun()
```

Output:

```
Welcome to Python
```

Defining and calling a function with parameters

```
def function_name(parameter: data_type) -> return_type:
   """Docstring"""
   # body of the function
   return expression
```

Example:

```
def add(num1: int, num2: int) -> int:
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

```
    """Add two numbers"""

    num3 = num1 + num2


    return num3


# Driver code

num1, num2 = 5, 15

ans = add(num1, num2)

print(f"The addition of {num1} and {num2} results {ans}.")
```

Output:

```
The addition of 5 and 15 results 20.
```

Types of Python Function Arguments

Python supports various types of arguments that can be passed at the time of the function call. In Python, we have the following 4 types of function arguments.

- **Default argument**
- **Keyword arguments (named arguments)**
- **Positional arguments**
- **Arbitrary arguments** (variable-length arguments *args and **kwargs)

Example:

```
# Python program to demonstrate

# default arguments

def myFun(x, y=50):

    print("x: ", x)

    print("y: ", y)
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

```
# Driver code (We call myFun() with only
# argument)
myFun(10)
```

Output:

```
x:  10
y:  50
```

Example:

```
# Python program to demonstrate Keyword Arguments
def student(firstname, lastname):
     print(firstname, lastname)


# Keyword arguments
student(firstname='Geeks', lastname='Practice')
student(lastname='Practice', firstname='Geeks')
```

Example:

```
def nameAge(name, age):
     print("Hi, I am", name)
     print("My age is ", age)


# You will get correct output because
# argument is given in order
print("Case-1:")
nameAge("Suraj", 27)
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

```
# You will get incorrect output because

# argument is not in order

print("\nCase-2:")

nameAge(27, "Suraj")
```

Output:

Case-1:

```
Hi, I am Suraj

My age is  27
```

Case-2:

```
Hi, I am 27

My age is  Suraj
```

Example:

```
# Python program to illustrate

# *args for variable number of arguments

def myFun(*argv):

     for arg in argv:

          print(arg)


myFun('Hello', 'Welcome', 'to', 'Python')
```

Output:

```
Hello

Welcome

to

Python
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

Recursive Functions in Python

Recursion in Python refers to when a function calls itself. There are many instances when you have to build a recursive function to solve **Mathematical and Recursive Problems.**

Using a recursive function should be done with caution, as a recursive function can become like a non-terminating loop. It is better to check your exit statement while creating a recursive function.

Example

```
def factorial(n):
     if n == 0:
          return 1
     else:
          return n * factorial(n - 1)


print(factorial(4))
```

Output:

```
24
```

Example:

```
def square_value(num):
     """This function returns the square
     value of the entered number"""
     return num**2



print(square_value(2))
print(square_value(-4))
```

Output:

```
4
```

Example:

```
# Here x is a new reference to same list lst
def myFun(x):
     x[0] = 20


# Driver Code (Note that lst is modified
# after function call.
lst = [10, 11, 12, 13, 14, 15]
myFun(lst)
print(lst)
```

Output:

```
[20, 11, 12, 13, 14, 15]
```

Python Decorators

In Python, a decorator is a design pattern that allows you to modify the functionality of a function by wrapping it in another function.

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

The outer function is called the decorator, which takes the original function as an argument and returns a modified version of it.

Prerequisites for learning decorators

1. A function is an object. Because of that, a function can be assigned to a variable. The function can be accessed from that variable.

```
def my_function():
    print('I am a function.')
```

```
# Assign the function to a variable without parenthesis. We don't want to execute the function.
description = my_function

# Accessing the function from the variable I assigned it to.

print(description())
```

 # Output

'I am a function.'

2. A function can be nested within another function.

```
def outer_function():

    def inner_function():

        print('I came from the inner function.')
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

```
    # Executing the inner function inside the outer
function.
    inner_function()

outer_function()

# Output

I came from the inner function.
```

Note that the **inner_function** is not available outside the outer_function. If I try to execute the inner_function outside of the outer_function I receive a NameError exception.

```
inner_function()

Traceback (most recent call last):
  File "/tmp/my_script.py", line 9, in <module>
    inner_function()

NameError: name 'inner_function' is not defined
```

3. Since a function can be nested inside another function it can also be returned.

```
def outer_function():
    '''Assign task to student'''

    task = 'Read Python book chapter 3.'
    def inner_function():
        print(task)
    return inner_function

homework = outer_function()
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

```
## call function
homework()

# Output

'Read Python book chapter 5.'
```

4. A function can be passed to another function as an argument.

```
def friendly_reminder(func):
    '''Reminder for friend'''

    func()
    print('Don\'t forget to bring your wallet!')

def action():

    print('I am going to the store buy you something
nice.')



# Calling the friendly_reminder function with the action
function used as an argument.

friendly_reminder(action)

# Output

I am going to the store buy you something nice.

Don't forget to bring your wallet!
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

A Python decorator is a function that takes in a function and returns it by adding some functionality.

In fact, any object which implements the special __call__() method is termed callable. So, in the most basic sense, a decorator is a callable that returns a callable.

Basically, a decorator takes in a function, adds some functionality and returns it.

```
def make_pretty(func):
    def inner():
        print("I got decorated")
        func()
    return inner


def ordinary():
    print("I am ordinary")



# Output: I am ordinary
```

Here, we have created two functions:

- ordinary() that prints "I am ordinary"

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

- make_pretty() that takes a function as its argument and has a nested function named inner(), and returns the inner function.

We are calling the ordinary() function normally, so we get the output "I am ordinary".

Now, let's call it using the decorator function.

```
def make_pretty(func):
    # define the inner function
    def inner():
        # add some additional behavior to decorated function
        print("I got decorated")

        # call original function
        func()
    # return the inner function
    return inner
# define ordinary function
def ordinary():
    print("I am ordinary")
# decorate the ordinary function
decorated_func = make_pretty(ordinary)
```

```
# call the decorated function

decorated_func()
```

##Output

I got decorated

I am ordinary

@ Symbol With Decorator

Instead of assigning the function call to a variable, Python provides a much more elegant way to achieve this functionality using the @ symbol. For example,

```
def make_pretty(func):
    def inner():
        print("I got decorated")
        func()
    return inner


@make_pretty
def ordinary():
    print("I am ordinary")


## make function call
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

```
ordinary()


##Output

I got decorated

I am ordinary
```

Here, the `ordinary()` function is decorated with the `make_pretty()` decorator using the `@make_pretty` syntax, which is equivalent to calling `ordinary = make_pretty(ordinary)`.

Decorating Functions with Parameters

```
def smart_divide(func):

    def inner(a, b):

        print("I am going to divide", a, "and", b)

        if b == 0:

            print("Whoops! cannot divide")

            return

        return func(a, b)
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

```
    return inner



@smart_divide

def divide(a, b):

    print(a/b)



divide(2,5)

divide(2,0)



## Output

I am going to divide 2 and 5

0.4

I am going to divide 2 and 0

Whoops! cannot divide
```

Here, when we call the **divide()** function with the arguments (2,5), the **inner()** function defined in the **smart_divide()** decorator is called instead.

This **inner()** function calls the original **divide()** function with the arguments 2 and 5 and returns the result, which is 0.4.

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

Similarly, When we call the **divide()** function with the arguments **(2,0),** the **inner()** function checks that b is equal to 0 and prints an error message before returning None.

QUESTIONS:

A. **Decision-making statements**

1. Write Python code that asks:x1,x2,y1,y2
   And Calculate the slope of the line slope=(y2-y1)/(x2-x1)
   - If the slope is positive, print "positive slope".
   - If the slope is negative, print "negative slope".
   - If the slope is zero, print "horizontal line".
   - If the denominator is 0, print "vertical line".

2. Use a while loop to display a table of Celsius temperatures from 20 to 10 and their Fahrenheit equivalents.
   The formula for converting a temperature from Celsius to Fahrenheit is:
   F=(9/5)*C+32

3. Write a Python program to create a list of integers with repetitions and display the list of pairs as follows:

   for each number n that appears in l, there should be exactly one pair (n,r) in the output list where r is the number of repetitions of n in l. The final list should be sorted in ascending order by r, the number of repetitions.
   For instance:
   INPUT: ([13,12,11,13,14,13,7,7,13,14,12])
   OUTPUT: [(11, 1), (7, 2), (12, 2), (14, 2), (13, 4)]

Hints:

1	Use for loop to iterate the list
2	Create a temporary dictionary or list any data structure to store answer

3	check if list items are already in the temporary dictionary/list or not ,maintain the occurrence accordingly.

B. Program using Functions

1. Write a function to check if a given number is prime or not.
2. Write a python program to calculate factorial value of numbers
   a) Without recursive function
   b) With Recursive function
4. Write a program using filter() to filter out even numbers from a list.
5. Write a program using map() to filter out even numbers from a list.

Questions for Write-up

1. **Difference between Recursive and Non-recursive Functions with**
2. **Different syntaxes of writing function decorators.**
3. **Explain use cases of python decorators.**
4. **Limitations of using Python Decorators**

OBSERVATIONS / DISCUSSION OF RESULT:

This section should interpret the outcome of the experiment. The observations can be visually represented using images, tables, graphs, etc. This section should answer the question "What do the result tell us?" Compare and interpret your results with expected behavior. Explain unexpected behavior, if any.

CONCLUSION:

Base all conclusions on your actual results; describe the meaning of the experiment and the implications of your results.

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)
DEPARTMENT OF INFORMATION TECHNOLOGY

REFERENCES:

Website References:

1. https://www.w3schools.com/python
2. https://www.tutorialspoint.com/
3. https://www.programiz.com/

Anish Sharma

60003220045

```
x1,x2,y1,y2 = map(int,input().split())
```

```
1 2 3 4
```

```
if x2-x1==0:
    print("Vertical line")
elif (y2-y1)/(x2-x1)>0:
    print("Postive slope")
elif (y2-y1)/(x2-x1)<0:
    print("Negative slope")
else:
    print("Horizontal line")
```

```
Postive slope
```

```
print("Celcius Fahrenheit")
for i in range(10,21):
    print(i,"     ",(9/5)*i+32)
```

```
Celcius Fahrenheit
10       50.0
11       51.8
12       53.6
13       55.400000000000006
14       57.2
15       59.0
16       60.8
17       62.6
18       64.4
19       66.2
20       68.0
```

```
lst =[13,12,11,13,14,13,7,7,13,14,12]
unique = set(lst)
d={}
temp=[]
for i in unique:
    r=0
    for j in lst:
        if i==j:
            r+=1
    temp.append([i,r])

temp
```

```
def sortSecond(val):
    return val[1]
temp.sort(key=sortSecond)
temp
```

```
[[11, 1], [7, 2], [12, 2], [14, 2], [13, 4]]
```

```
temp.sort(key=sortSecond, reverse=True)
temp
```

```
[[13, 4], [7, 2], [12, 2], [14, 2], [11, 1]]
```

```
n=5
flag=0
if n==1 or n==0:
    print("Not a prime")
elif n==2:
    print("Prime")
else:
    for i in range(2,n):
        if n%i==0:
            flag=1
            break
if flag==1:
    print("Not prime")
else:
    print("Prime")
```

```
Prime
```

```
flag=0
n=90
if n==1 or n==0:
    print("Not a prime")
elif n==2:
    print("Prime")
else:
    for i in range(2,n):
        if n%i==0:
            flag=1
            break
if flag==1:
    print("Not prime")
else:
    print("Prime")
```

```
Not prime
```

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

```
120
```

```
n=5
fact=1
for i in range(1,6):
    fact = fact*i
fact
```

```
120
```

```
lst = [1,2,3,4,5,6,7,8,9,10]

result = filter(lambda x: x % 2== 0, lst)
print(list(result))
```

```
[2, 4, 6, 8, 10]
```

```
lst = [1,2,3,4,5,6,7,8,9,10]

result = map(lambda x: x % 2== 0, lst)
print(list(result))
```

```
[False, True, False, True, False, True, False, True, False, True]
```

```
def check(n):
    return n%2==0
numbers = [1, 2, 3, 4]
result = map(check, numbers)
print(list(result))
```

```
[False, True, False, True]
```