CH:3 FUNCTIONS SCOPING AND ABSTARCTION(5 M)

Why functions?

- reusability
- consise code
- modularity

Defining functions in py

In [ ]:
```
def greet():
    print("Hello")
```

Calling functions

In [ ]:
```
greet() #Hello
print(greet()) #None as greet method return type is none
```

Function specification

- It specifies the function name,parameter and the return type

In [ ]:
```
def add(x,y):# here x and y are PARAMETERS
    """
    This function return addition of x and y
    Parametres:
    x(int) : 1st number
    y(int) : 2nd number
    Returns:
    int: the addition of x and y
    """
    return x+y
```

In [ ]:
```
add(4,5) # here 4 and 5 are ARGUMENTS
```

In [ ]:
```
add??
```

Types of functions

- Inbuilt functions

In [ ]:
```
dir(__builtins__) # a list containing Builtin functions
```

User Defined functions

1. No parameter no return

In [ ]:
```
def printline():
    print("Hello world")

printline()
```

2.No parameter with return

In [ ]:
```
def printline():
    return "Hello world"

printline()
```

3.With parameter no return

In [ ]:
```
def printline(s):
    print(m)

m= input("enter greeting")
printline(m)

```

4.with parameter with return

In [ ]:
```
def printline(s):
    return s
    print("Will this line run")

m= input("enter greeting")
printline(m)

```

Returning multiple values

In [ ]:
```
def sumsub(x,y):
    sum = x+y
    sub = x-y
    return sum,sub
m=sumsub(3,1)
print(type(m))
```

In [ ]:
```
m,n = sumsub(3,1)
print(m)
print(n)
print(type(m))
print(type(n))
```

WAP that print addition sub division and multipication of 2 number

In [ ]:
```
def cal(x,y):

    print("Addition is",x+y)
    print("Subst is",x-y)
    print("Multi is",x*y)
    print("Divis is",x/y)
```

In [ ]:
```
cal(3,4)
```

Parameter and argumnets

- The value in paranthesis used while defining a function are called paramters
- The value passed while calling a function are called arguments

Types of Arguments

Default arguments

In [ ]:
```
def square(x=20):
    return x*x
print(square())
print(square(10))
```

Position arguments

- The number or argument and thier posiotn must match
- if we change the order of argument the result will way

- if we change the number of arguments we will get error

In [ ]:
```
def sub(x,y):
    return x-y
print(sub(5,6))
print(sub(6,5))
print(sub(5,6,7)) #error
```

Keyword argument

- in case of all keyword argument the order doesnt matter
- one can use combination of keyword and postional argument
- keyword argument always follow posiotnal argument

In [ ]:
```
def wish(name,msg):
    print("hello",name,msg)
```

In [ ]:
```
wish(name='python',msg='good morning')
```

In [ ]:
```
wish(msg='good morning',name='java')
```

In [ ]:
```
wish("C++",msg="good afternoon")
```

In [ ]:
```
wish("C++",name="good aftrenoon") # error
wish(msg="good afternoon","C++") # error
```

Variable length argument

In [ ]:
```
def sum(*n):
    total = 0
    for i in n:
        total+=i
    print("the sum is",total)
```

In [ ]:
```
sum(10)
```

In [ ]:
```
sum(10,20)
```

In [ ]:
```
sum(10,-10,10,20,30)
```

Function scope

Local varibale

- a local varibale is declared inside function has started executon and are lost when the function terminates

In [ ]:
```
x=5
def fun():
    x=100
    print(x)
```

In [ ]:
```
fun()
```

In [ ]:
```
print(x)
```

Global keyword

In [ ]:
```
x=5
def fun2():
    global x
    x=100
    print(x)
```

In [ ]:
```
fun2()
```

Scoping rule

- legb rule

1. Local
2. Enclosed
3. Global
4. Builtin

Nested functions

In [ ]:
```
def f():
    def g():
        print("inside g function")
    g()
    print("inside f functions")
```

In [ ]:
```
f()
```

In [ ]:
```
def g(x):
    def h():
        x='abc'
        return x
    x = x+1
    print("in g function x is",x)
    print(h())
    return x
```

In [ ]:
```
x = 3
z = g(x)
print(z)
```

In [ ]:
```
whos # it is used to see the datatypes declared
```

Duplicate function

In [ ]:
```
def add(x,y):
    return x+y
def add(x,y):
    return x-y
```

In [ ]:
```
add(2,3)
```

WAP to swap

In [ ]:
```
x = 123456789
y= x%10
digit=0
print(y)
z=x//100000
print(z)
between = int((x%100000)/10)
print(between)
print(f"{y}{between}{z}")
while x!=0:
    digit=digit+1
    x=x//10
print("digit is",digit)

```

In [ ]:
```

```

In [ ]:
```

```