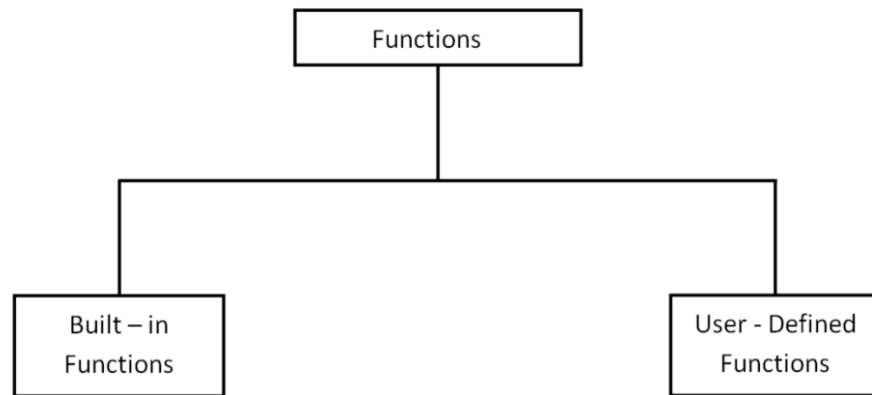


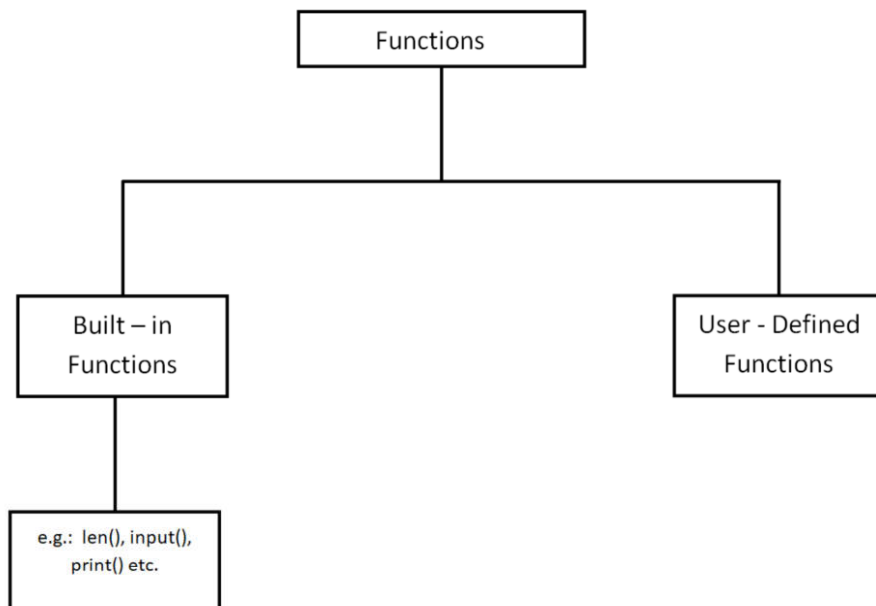
Functions



User-Defined Function

- ☐ A **function** is a block of statements that perform a specific task.
- ☐ **Functions** allow us to break our code into **manageable, bite-sized chunks**.
- ☐ Functions are used to utilize the code in more than one place in a program.
- ☐ Use of functions increases program readability.
- ☐ Use of functions reduces the chances of error.
- ☐ Program modification becomes easier by using functions.

Functions



User-Defined Function

Syntax

```
def function_name(parameters):  
    """documantion string"""  
    statement(s)  
    return result(s)
```

- ☐ **parameters**(optional): can be one or more.
- ☐ **return**(optional): statement ends the execution of the function call and returns the results (one or more).
- ☐ If there is no return statement in the function body, the function ends, when the control flow reaches the end of the function body and the value "None" will be returned.

Example

```
#print using function
def display():
    """Display function"""
    print("Welcome to IC152 Class")

display()
print("Function documentation:",display.__doc__)
```

Output:

```
Welcome to IC152 class
Function documentation: Display Function
```

Example

```
# function to calculate area and perimeter of circle
def circle(r):
    area = 3.14 * r * r
    perimeter = 2 * 3.14 * r
    return area, perimeter

radius = 5
print("Function documentation:",circle.__doc__)
area,perimeter = circle(radius)
print("Area of circle is {0:.2f} and perimeter is
      {1:.2f}".format(area,perimeter))
result = circle(radius)
print(result)
```

```
Function documentation: None
Area of circle is 78.50 and perimeter is 31.40
(78.5, 31.400000000000002)
```

Default Parameter Value

```
# function to calculate area and perimeter of circle
def circle(r = 3):
    area = 3.14 * r * r
    perimeter = 2 * 3.14 * r
    return area, perimeter

radius = 5
print("Function documentation:",circle.__doc__)
Area,Perimeter = circle()
print("Area of circle is {0:.2f} and perimeter is
      {1:.2f}".format(Area,Perimeter))
result = circle(radius)
print(result)
```

```
Function documentation: None
Area of circle is 28.26 and perimeter is 18.84
(78.5, 31.400000000000002)
```

Default Parameter Value

Any number of arguments in a function can have a default value. But once we have a default argument, all the arguments to its right also must have default values.

```
def fun(a , b = 10 ):
    print("a :",a)
    print("b :",b)
```

```
a = 5
b = 20
fun(a)
```

```
a: 5
b: 10
```

```
def fun(a = 10, b):
    print("a :",a)
    print("b :",b)
```

```
a = 5
b = 20
fun(a,b)
```

```
SyntaxError: non-default
argument follows default
argument
```

Keyword Arguments

In keyword arguments we pass arguments as **key = value**. Order of arguments does not matter in keyword argument method.

```
def fun(a = 5, b = 10):  
    print("a :",a)  
    print("b :",b)
```

```
fun(b = 20)
```

```
a: 5  
b: 20
```

```
def fun(a, b):  
    print("a :",a)  
    print("b :",b)
```

```
fun(b = 20, a = 5)
```

```
a: 5  
b: 20
```

Lifetime and Scope of Variables

```
def func():  
    a = 20  
    b = 10  
    print("Value of a inside function is",a)  
  
a = 30  
func()  
print("Value of a outside function is",a)  
print("Value of b outside function is",b)
```

Output

```
Value of a inside function is 20  
Value of a outside function is 30  
NameError: name 'b' is not defined
```

Lifetime and Scope of Variables

- ☐ Lifetime of a variable is the period throughout which the variable exists in the memory.
- ☐ The lifetime of variables inside a function is as long as the function executes.
- ☐ Scope of a variable is the portion of a program where the variable is recognized.
- ☐ Parameters and variables defined inside a function are not visible from outside. Hence, they have a local scope.

Python Anonymous/Lambda Function

- ☐ **lambda** keyword is used to create anonymous functions.
- ☐ Lambda Function can have any number of arguments but only one expression, which is evaluated and returned.
- ☐ **lambda functions** are used when we require a nameless function for a short period of time.

Syntax

```
lambda arguments: expression
```

Python Anonymous/Lambda Function

```
>>>cube = lambda a: a*a*a
>>>print(cube(5))
125
```

Use:

```
>>>old_list = [1, 2, 3, 4, 5]
>>>new_list = list(map(lambda a: a*a, old_list))
>>>print(new_list)
[1, 4, 9, 16, 25]
```

```
def function(y):
    print("Id inside function before assignment:", id(y))
    y = 10
    print("Id inside function after assignment:", id(y))
    print("Value inside function:", y)
```

```
x = 5
print("Id outside function before function call:", id(x))
function(x)
print("Id outside function after function call:", id(x))
print("Value outside function:", x)
```

```
Id outside function before function call: 140715184067520
Id inside function before assignment: 140715184067520
Id inside function after assignment: 140715184067680
Value inside function: 10
Id outside function after function call: 140715184067520
Value outside function: 5
```

```
def function(x):
    print("Id inside function before assignment:", id(x))
    x = 10
    print("Id inside function after assignment:", id(x))
    print("Value inside function:", x)
```

```
x = 5
print("Id outside function before function call:", id(x))
function(x)
print("Id outside function after function call:", id(x))
print("Value outside function:", x)
```

```
Id outside function before function call: 140715184067520
Id inside function before assignment: 140715184067520
Id inside function after assignment: 140715184067680
Value inside function: 10
Id outside function after function call: 140715184067520
Value outside function: 5
```

```
def function(x):
    print("Id inside function before assignment:", id(x))
    x = 5
    print("Id inside function after assignment:", id(x))
    print("Value inside function:", x)
```

```
x = 5
print("Id outside function before function call:", id(x))
function(x)
print("Id outside function after function call:", id(x))
print("Value outside function:", x)
```

```
Id outside function before function call: 140715184067520
Id inside function before assignment: 140715184067520
Id inside function after assignment: 140715184067520
Value inside function: 5
Id outside function after function call: 140715184067520
Value outside function: 5
```


User-Defined Function

- ☐ A **function** is a block of statements that perform a specific task.
- ☐ **Functions** allow us to break our code into **manageable, bite-sized chunks**.
- ☐ Functions are used to utilize the code in more than one place in a program.
- ☐ Use of functions increases program readability.
- ☐ Use of functions reduces the chances of error.

Example

```
#print using function
def display():
    """Display function"""
    print("Welcome to IC152 Class")

display()
print("Function documentation:",display.__doc__)
```

Output:

User-Defined Function

Syntax

```
def function_name(parameters):
    """documantion string"""
    statement(s)
    return result(s)
```

- ☐ **parameters**(optional): can be one or more.
- ☐ **return**(optional): statement ends the execution of the function call and returns the results (one or more).

Example

```
# function to calculate area and perimeter of circle
def circle(r):
    area = 3.14 * r * r
    perimeter = 2 * 3.14 * r
    return area, perimeter

radius = 5
print("Function documentation:",circle.__doc__)
area,perimeter = circle(radius)
print("Area of circle is {0:.2f} and perimeter is {1:.2f}".format(area,perimeter))
result = circle(radius)
print(result)
```

Default Parameter Value

```
# function to calculate area and perimeter of circle
def circle(r = 3):
    area = 3.14 * r * r
    perimeter = 2 * 3.14 * r
    return area, perimeter

radius = 5
print("Function documentation:", circle.__doc__)
Area, Perimeter = circle()
print("Area of circle is {0:.2f} and perimeter is
      {1:.2f}".format(Area, Perimeter))
result = circle(radius)
print(result)
```

Keyword Arguments

In keyword arguments we pass arguments as **key = value**. Order of arguments does not matter in keyword argument method.

```
def fun(a = 5, b = 10):
    print("a :", a)
    print("b :", b)

fun(b = 20)
```

```
def fun(a, b):
    print("a :", a)
    print("b :", b)

fun(b = 20, a = 5)
```

```
a: 5
b: 20
```

Default Parameter Value

Any number of arguments in a function can have a default value. But once we have a default argument, all the arguments to its right also must have default values.

```
def fun(a, b = 10):
    print("a :", a)
    print("b :", b)
```

```
a = 5
b = 20
fun(a)
```

```
def fun(a = 10, b):
    print("a :", a)
    print("b :", b)
```

```
a = 5
b = 20
fun(a, b)
```

```
a: 5
b: 10
```

Lifetime and Scope of Variables

- ☐ Lifetime of a variable is the period throughout which the variable exists in the memory.
- ☐ The lifetime of variables inside a function is as long as the function executes.
- ☐ Scope of a variable is the portion of a program where the variable is recognized.

Lifetime and Scope of Variables

```
def func():  
    a = 20  
    b = 10  
    print("Value of a inside function is",a)  
  
a = 30  
func()  
print("Value of a outside function is",a)  
print("Value of b outside function is",b)
```

Python Anonymous/Lambda Function

```
>>> cube = lambda a: a*a*a  
>>> print(cube(5))  
125
```

Use:

```
>>> old_list = [1, 2, 3, 4, 5]  
>>> new_list = list(map(lambda a: a*a, old_list))  
>>> print(new_list)
```

Lifetime and Scope of Variables

```
def func():  
    a = 20  
    b = 10  
    print("Value of a inside function is",a)  
  
a = 30  
func()  
print("Value of a outside function is",a)  
print("Value of b outside function is",b)
```

Output

```
def function(x):  
    print("Id inside function before assignment:", id(x))  
    x = 10  
    print("Id inside function after assignment:", id(x))  
    print("Value inside function:", x)  
  
x = 5  
print("Id outside function before function call:", id(x))  
function(x)  
print("Id outside function after function call:", id(x))  
print("Value outside function:", x)
```

```
Id outside function before function call: 140715184067520  
Id inside function before assignment: 140715184067520  
Id inside function after assignment: 140715184067680  
Value inside function: 10  
Id outside function after function call: 140715184067520
```

```
def function(y):
    print("Id inside function before assignment:", id(y))
    y = 10
    print("Id inside function after assignment:", id(y))
    print("Value inside funtion:", y)

x = 5
print("Id ouside function before function call:", id(x))
function(x)
print("Id ouside function after function call:", id(x))
print("Value ouside function:", x)
```

```
Id ouside function before function call: 140715184067520
Id inside function before assignment: 140715184067520
Id inside function after assignment: 140715184067680
Value inside funtion: 10
Id ouside function after function call: 140715184067520
```

```
def fun(my_str):
    print("Id inside fun before append:", id(my_str))
    my_str + " world"
    print("Id inside fun after append:", id(my_str))
    print("my_str inside fun:", my_str)
my_str = "Hello"
print("Id outside fun before fun call:", id(my_str))
print("my_str outside fun before fun call:", my_str)
fun(my_str)
print("Id outside after fun call:", id(my_str))
print("my_str outside after fun call:", my_str)
```

```
Id outside fun before fun call: 2013692429848
my_str outside fun before fun call: Hello
Id inside fun before append: 2013692429848
Id inside fun after append: 2013692429848
my_str inside fun: Hello
Id outside after fun call: 2013692429848
my_str outside after fun call: Hello
```

```
def function(x):
    print("Id inside function before assignment:", id(x))
    x = 5
    print("Id inside function after assignment:", id(x))
    print("Value inside funtion:", x)

x = 5
print("Id ouside function before function call:", id(x))
function(x)
print("Id ouside function after function call:", id(x))
print("Value ouside function:", x)
```

```
Id ouside function before function call: 140715184067520
Id inside function before assignment: 140715184067520
Id inside function after assignment: 140715184067520
Value inside funtion: 5
Id ouside function after function call: 140715184067520
```

```
def fun(my_str):
    print("Id inside fun before append:", id(my_str))
    my_str = my_str + " world"
    print("Id inside fun after append:", id(my_str))
    print("my_str inside fun:", my_str)
my_str = "Hello"
print("Id outside fun before fun call:", id(my_str))
print("my_str outside fun before fun call:", my_str)
fun(my_str)
print("Id outside after fun call:", id(my_str))
print("my_str outside after fun call:", my_str)
```

```
Id outside fun before fun call: 2013692429848
my_str outside fun before fun call: Hello
Id inside fun before append: 2013692429848
Id inside fun after append: 2013692619824
my_str inside fun: Hello world
Id outside after fun call: 2013692429848
my_str outside after fun call: Hello
```



```
def fun(my_list):
    print("Id inside fun before append:", id(my_list))
    my_list.append(9)
    print("Id inside fun after append:", id(my_list))
    print("my_list inside fun:", my_list)
my_list = [5,7,8]
print("Id ouside fun before fun call:", id(my_list))
print("my_list outside fun before fun call:", my_list)
fun(my_list)
print("Id ouside fun after fun call:", id(my_list))
print("my_list ouside fun after fun call:", my_list)
```

```
Id ouside fun before fun call: 2013692463368
my_list outside fun before fun call: [5, 7, 8]
Id inside fun before append: 2013692463368
Id inside fun after append: 2013692463368
my_list inside fun: [5, 7, 8, 9]
Id ouside fun after fun call: 2013692463368
my_list ouside fun after fun call: [5, 7, 8, 9]
```

```
def fun(my_list):
    print("my_list inside fun before assignment", my_list)
    print("Id inside fun before assignment:", id(my_list))
    my_list = [1,'IC152',2,3]
    print("Id inside fun after assignment:", id(my_list))
    print("my_list inside fun:", my_list)
my_list = [5,7,8]
print("Id ouside fun before fun call:", id(my_list))
print("my_list outside fun before fun call:", my_list)
fun(my_list)
print("Id ouside fun after fun call:", id(my_list))
print("my_list ouside fun after fun call:", my_list)
```

```
Id ouside fun before fun call: 2013692514824
my_list outside fun before fun call: [5, 7, 8]
my_list inside fun before assignment [5, 7, 8]
Id inside fun before assignment: 2013692514824
Id inside fun after assignment: 2013692617992
my_list inside fun: [1, 'IC152', 2, 3]
Id ouside fun after fun call: 2013692514824
my_list ouside fun after fun call: [5, 7, 8]
```

```
def fun(my_list1):
    print("my_list1 before append", my_list1)
    print("my_list1 Id before append:", id(my_list1))
    my_list1.append(9)
    print("my_list1 Id after append:", id(my_list1))
    print("my_list1 inside fun:", my_list1)
my_list = [5,7,8]
print("my_list Id before fun call:", id(my_list))
print("my_list before fun call:", my_list)
fun(my_list)
print("my_list Id after fun call:", id(my_list))
print("my_list after fun call:", my_list)
print("my_list1 outside fun:", my_list1)
```

```
my_list Id before fun call: 2013692513544
my_list before fun call: [5, 7, 8]
my_list1 before append [5, 7, 8]
my_list1 Id before append: 2013692513544
my_list1 Id after append: 2013692513544
my_list1 inside fun: [5, 7, 8, 9]
my_list Id after fun call: 2013692513544
my_list after fun call: [5, 7, 8, 9]
NameError: name 'my_list1' is not defined
```

Home Work

? Questions:

- 1 Do the same exercise with list for different kind of list methods?

Home Work

? Questions:

- 1 Do the same exercise with list for different kind of list methods?
- 2 Pass tuple or set to function and apply tuple or string methods inside function and figure out what will happen?

Variable Length Argument

```
def Districts(MyDistrict, *Neighbour):
    print("My district is " + MyDistrict)

    if len(Neighbour) > 0:
        for i in range(len(Neighbour)):
            print("Neighbour {:2d} is {}".format(i+1,
                                                    Neighbour[i]))

Neighbouring = ("Bilaspur", "Hamirpur", "Kangra", "Kullu",
                "Shimla")
Districts("Mandi", Neighbouring)
```

```
My district is Mandi
Neighbour 1 is ('Bilaspur', 'Hamirpur', 'Kangra', 'Kullu',
               'Shimla').
```

Variable Length Argument

```
def Districts(MyDistrict, *Neighbour):
    print("My district is " + MyDistrict)

    if len(Neighbour) > 0:
        for i in range(len(Neighbour)):
            print("Neighbour {:2d} is {}".format(i+1,
                                                    Neighbour[i]))

Districts("Mandi")
Districts("Mandi", "Bilaspur", "Hamirpur", "Kangra",
          "Kullu", "Shimla")
```

```
My district is Mandi
My district is Mandi
Neighbour 1 is Bilaspur.
Neighbour 2 is Hamirpur.
Neighbour 3 is Kangra.
Neighbour 4 is Kullu.
Neighbour 5 is Shimla.
```

Variable Length Argument

```
def Districts(MyDistrict, *Neighbour):
    print("My district is " + MyDistrict)

    if len(Neighbour) > 0:
        for i in range(len(Neighbour)):
            print("Neighbour {:2d} is {}".format(i+1,
                                                    Neighbour[i]))

Neighbouring = ("Bilaspur", "Hamirpur", "Kangra", "Kullu",
                "Shimla")
Districts("Mandi", *Neighbouring)
```

```
My district is Mandi
Neighbour 1 is Bilaspur.
Neighbour 2 is Hamirpur.
Neighbour 3 is Kangra.
Neighbour 4 is Kullu.
Neighbour 5 is Shimla.
```

Variable Length Argument

```
def Districts(MyDistrict, *Neighbour):
    print("My district is " + MyDistrict)

    if len(Neighbour) > 0:
        for i in range(len(Neighbour)):
            print("Neighbour {:2d} is {}".format(i+1,
                Neighbour[i]))

Neighbouring = ("Bilaspur", "Hamirpur", "Kangra", "Kullu",
    "Shimla")
Districts(*Neighbouring)
```

```
My district is Bilaspur
Neighbour  1 is Hamirpur.
Neighbour  2 is Kangra.
Neighbour  3 is Kullu.
Neighbour  4 is Shimla.
```

Variable Length Argument

```
def fun(**course):
    print("-----")
    print(type(course))
    for key, value in course.items():
        print ("%s : %s" %(key, value))
    print("-----")

fun(IC150 = "Computing", IC110 = "Mathematics", IC160 =
    "Electrical")
```

```
-----
<class 'dict'>
IC150 : Computing
IC110 : Mathematics
IC160 : Electrical
-----
```

Variable Length Argument

```
def Mean(*x):
    S = 0
    for i in range(len(x)):
        S += x[i]
    if len(x) > 0:
        S = S/len(x)
    print ("The mean is ", S)
    return S
```

```
Mean()
Mean(10)
Mean(67, 52, 70, 69, 86)
Marks = [67, 52, 70, 69, 86]
Mean(*Marks)
Mean(Marks)
```

```
The mean is  0
The mean is  10.0
The mean is  68.8
The mean is  68.8
Error
```

Variable Length Argument

```
def fun(**course):
    print("-----")
    print(type(course))
    for key, value in course.items():
        print ("%s : %s" %(key, value))
    print("-----")

CourseDic = {"IC150" : "Computing", "IC110" :
    "Mathematics", "IC160" : "Electrical"}
fun(**CourseDic)
```

```
-----
<class 'dict'>
IC150 : Computing
IC110 : Mathematics
IC160 : Electrical
-----
```


Function Call

```
def fun():  
    print(S, id(S))  
  
S = "Data Science"  
print(S, id(S))  
fun()  
print(S, id(S))
```

```
Data Science 2139013498352  
Data Science 2139013498352  
Data Science 2139013498352
```

global Variables

```
def fun():  
    global S  
    print(S, id(S))  
    S = "Data Scinece & Engineering"  
    print(S, id(S))  
  
S = "Data Science"  
print(S, id(S))  
fun()  
print(S, id(S))
```

global Variables

```
def fun():  
    global S  
    print(S, id(S))  
    S = "Data Scinece & Engineering"  
    print(S, id(S))  
  
S = "Data Science"  
print(S, id(S))  
fun()  
print(S, id(S))
```

global Variables

```
def fun():  
    global S  
    print(S, id(S))  
    S = "Data Scinece & Engineering"  
    print(S, id(S))  
  
S = "Data Science"  
print(S, id(S))  
fun()  
print(S, id(S))
```

```
Data Science 2139013500080
```


global Variables

```
def fun():
    global S
    print(S, id(S))
    S = "Data Scinece & Engineering"
    print(S, id(S))

S = "Data Science"
print(S, id(S))
fun()
print(S, id(S))
```

```
Data Science 2139013500080
Data Science 2139013500080
```

global Variables

```
def fun():
    global S
    print(S, id(S))
    S = "Data Scinece & Engineering"
    print(S, id(S))

S = "Data Science"
print(S, id(S))
fun()
print(S, id(S))
```

```
Data Science 2139013500080
Data Science 2139013500080
Data Scinece & Engineering 2139013437360
Data Scinece & Engineering 2139013437360
```

global Variables

```
def fun():
    global S
    print(S, id(S))
    S = "Data Scinece & Engineering"
    print(S, id(S))

S = "Data Science"
print(S, id(S))
fun()
print(S, id(S))
```

```
Data Science 2139013500080
Data Science 2139013500080
Data Scinece & Engineering 2139013437360
```

global Variables

```
def fun():
    global S
    print(S, id(S))
    S = S + " & Engineering"
    print(S, id(S))

S = "Data Science"
print(S, id(S))
fun()
print(S, id(S))
```

```
Data Science 2139013317616
Data Science 2139013317616
Data Science & Engineering 2139013437360
Data Science & Engineering 2139013437360
```

nonlocal Variables

The `nonlocal` bindings can only be used inside of nested functions.

```
def fun1():
    S = "IC150"
    def fun2():
        nonlocal S
        S = "IC152"

    print("Before calling fun2:", S)
    fun2()
    print("After calling fun2:", S)

S = "IC"
print("After assignment:", S)
fun1()
print("S in main:", S)
```

nonlocal Variables

The `nonlocal` bindings can only be used inside of nested functions.

```
def fun1():
    S = "IC150"
    def fun2():
        nonlocal S
        S = "IC152"

    print("Before calling fun2:", S)
    fun2()
    print("After calling fun2:", S)

S = "IC"
print("After assignment:", S)
fun1()
print("S in main:", S)
```

```
After assignment: IC
```

nonlocal Variables

The `nonlocal` bindings can only be used inside of nested functions.

```
def fun1():
    S = "IC150"
    def fun2():
        nonlocal S
        S = "IC152"

    print("Before calling fun2:", S)
    fun2()
    print("After calling fun2:", S)

S = "IC"
print("After assignment:", S)
fun1()
print("S in main:", S)
```

nonlocal Variables

The `nonlocal` bindings can only be used inside of nested functions.

```
def fun1():
    S = "IC150"
    def fun2():
        nonlocal S
        S = "IC152"

    print("Before calling fun2:", S)
    fun2()
    print("After calling fun2:", S)

S = "IC"
print("After assignment:", S)
fun1()
print("S in main:", S)
```

```
After assignment: IC
Before calling fun2: IC150
```

nonlocal Variables

The `nonlocal` bindings can only be used inside of nested functions.

```
def fun1():
    S = "IC150"
    def fun2():
        nonlocal S
        S = "IC152"

    print("Before calling fun2:", S)
    fun2()
    print("After calling fun2:", S)

S = "IC"
print("After assignment:", S)
fun1()
print("S in main:", S)
```

```
After assignment: IC
Before calling fun2: IC150
After calling fun2: IC152
```

nonlocal Variables

- ❑ A `nonlocal` variable has to be defined in the enclosing function scope.
- ❑ If the variable is **not** defined in the enclosing function scope, the variable cannot be defined in the nested scope.

```
def fun1():
    #S = "IC1502"
    def fun2():
        nonlocal S
        S = "IC152"

    print("Before calling fun2:", S)
    fun2()
    print("After calling fun2:", S)

S = "IC"
print("After assignment:", S)
fun1()
print("S in main:", S)
```

```
SyntaxError: no binding for nonlocal 'S' found
```

nonlocal Variables

The `nonlocal` bindings can only be used inside of nested functions.

```
def fun1():
    S = "IC150"
    def fun2():
        nonlocal S
        S = "IC152"

    print("Before calling fun2:", S)
    fun2()
    print("After calling fun2:", S)

S = "IC"
print("After assignment:", S)
fun1()
print("S in main:", S)
```

```
After assignment: IC
Before calling fun2: IC150
After calling fun2: IC152
S in main: IC
```

global Variables

```
def fun1():
    #S = "IC150"
    def fun2():
        global S
        S = "IC152"

    print("Before calling fun2:", S)
    fun2()
    print("After calling fun2:", S)

S = "IC"
print("After assignment:", S)
fun1()
print("S in main:", S)
```

global Variables

```
def fun1():
    #S = "IC150"
    def fun2():
        global S
        S = "IC152"

    print("Before calling fun2:", S)
    fun2()
    print("After calling fun2:", S)

S = "IC"
print("After assignment:", S)
fun1()
print("S in main:", S)
```

global Variables

```
def fun1():
    #S = "IC150"
    def fun2():
        global S
        S = "IC152"

    print("Before calling fun2:", S)
    fun2()
    print("After calling fun2:", S)

S = "IC"
print("After assignment:", S)
fun1()
print("S in main:", S)
```

```
After assignment: IC
Before calling fun2: IC
```

global Variables

```
def fun1():
    #S = "IC150"
    def fun2():
        global S
        S = "IC152"

    print("Before calling fun2:", S)
    fun2()
    print("After calling fun2:", S)

S = "IC"
print("After assignment:", S)
fun1()
print("S in main:", S)
```

```
After assignment: IC
```

global Variables

```
def fun1():
    #S = "IC150"
    def fun2():
        global S
        S = "IC152"

    print("Before calling fun2:", S)
    fun2()
    print("After calling fun2:", S)

S = "IC"
print("After assignment:", S)
fun1()
print("S in main:", S)
```

```
After assignment: IC
Before calling fun2: IC
After calling fun2: IC152
```


global Variables

```
def fun1():
    #S = "IC150"
    def fun2():
        global S
        S = "IC152"

    print("Before calling fun2:", S)
    fun2()
    print("After calling fun2:", S)

S = "IC"
print("After assignment:", S)
fun1()
print("S in main:", S)
```

```
After assignment: IC
Before calling fun2: IC
After calling fun2: IC152
S in main: IC152
```

Recursion

global Variables

```
def fun1():
    S = "IC150"
    def fun2():
        global S
        S = "IC152"

    print("Before calling fun2:", S)
    fun2()
    print("After calling fun2:", S)

fun1()
print("S in main:", S)
```

```
Before calling fun2: IC150
After calling fun2: IC150
S in main: IC152
```

```
def Summation(4):
    print("Calling summation with n = " + str(4))
    if n == 1:
        return 1
    else:
        Sum = 4**2 + Summation(4-1)
        return Sum

print(Summation(4))
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
  
print(Summation(4))
```


```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
  
print(Summation(4))
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
  
print(Summation(4))
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
  
print(Summation(4))
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
print(Summation(4))
```


```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
print(Summation(4))
```



```
def Summation(3):  
    print("Calling summation with n = " + str(3))  
    if n == 1:  
        return 1  
    else:  
        Sum = 3**2 + Summation(3-1)  
        return Sum
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
print(Summation(4))
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
print(Summation(4))
```



```
def Summation(3):  
    print("Calling summation with n = " + str(3))  
    if n == 1:  
        return 1  
    else:  
        Sum = 3**2 + Summation(3-1)  
        return Sum
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
print(Summation(4))
```

```
def Summation(3):  
    print("Calling summation with n = " + str(3))  
    if n == 1:  
        return 1  
    else:  
        Sum = 3**2 + Summation(3-1)  
        return Sum
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
print(Summation(4))
```

```
def Summation(3):  
    print("Calling summation with n = " + str(3))  
    if n == 1:  
        return 1  
    else:  
        Sum = 3**2 + Summation(3-1)  
        return Sum
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
print(Summation(4))
```

```
def Summation(3):  
    print("Calling summation with n = " + str(3))  
    if n == 1:  
        return 1  
    else:  
        Sum = 3**2 + Summation(3-1)  
        return Sum
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
print(Summation(4))
```

```
def Summation(3):  
    print("Calling summation with n = " + str(3))  
    if n == 1:  
        return 1  
    else:  
        Sum = 3**2 + Summation(3-1)  
        return Sum
```

```
def Summation(2):  
    print("Calling summation with n = " + str(2))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(2-1)  
        return Sum
```



```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
print(Summation(4))
```

```
def Summation(3):  
    print("Calling summation with n = " + str(3))  
    if n == 1:  
        return 1  
    else:  
        Sum = 3**2 + Summation(3-1)  
        return Sum
```

```
def Summation(2):  
    print("Calling summation with n = " + str(2))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(2-1)  
        return Sum
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
print(Summation(4))
```

```
def Summation(3):  
    print("Calling summation with n = " + str(3))  
    if n == 1:  
        return 1  
    else:  
        Sum = 3**2 + Summation(3-1)  
        return Sum
```

```
def Summation(2):  
    print("Calling summation with n = " + str(2))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(2-1)  
        return Sum
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
print(Summation(4))
```

```
def Summation(3):  
    print("Calling summation with n = " + str(3))  
    if n == 1:  
        return 1  
    else:  
        Sum = 3**2 + Summation(3-1)  
        return Sum
```

```
def Summation(2):  
    print("Calling summation with n = " + str(2))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(2-1)  
        return Sum
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
print(Summation(4))
```

```
def Summation(3):  
    print("Calling summation with n = " + str(3))  
    if n == 1:  
        return 1  
    else:  
        Sum = 3**2 + Summation(3-1)  
        return Sum
```

```
def Summation(2):  
    print("Calling summation with n = " + str(2))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(2-1)  
        return Sum
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
print(Summation(4))
```

```
def Summation(3):  
    print("Calling summation with n = " + str(3))  
    if n == 1:  
        return 1  
    else:  
        Sum = 3**2 + Summation(3-1)  
        return Sum
```

```
def Summation(2):  
    print("Calling summation with n = " + str(2))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(2-1)  
        return Sum
```

```
def Summation(1):  
    print("Calling summation with n = " + str(1))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(1-1)  
        return Sum
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
print(Summation(4))
```

```
def Summation(3):  
    print("Calling summation with n = " + str(3))  
    if n == 1:  
        return 1  
    else:  
        Sum = 3**2 + Summation(3-1)  
        return Sum
```

```
def Summation(2):  
    print("Calling summation with n = " + str(2))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(2-1)  
        return Sum
```

```
def Summation(1):  
    print("Calling summation with n = " + str(1))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(1-1)  
        return Sum
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
print(Summation(4))
```

```
def Summation(3):  
    print("Calling summation with n = " + str(3))  
    if n == 1:  
        return 1  
    else:  
        Sum = 3**2 + Summation(3-1)  
        return Sum
```

```
def Summation(2):  
    print("Calling summation with n = " + str(2))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(2-1)  
        return Sum
```

```
def Summation(1):  
    print("Calling summation with n = " + str(1))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(1-1)  
        return Sum
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
print(Summation(4))
```

```
def Summation(3):  
    print("Calling summation with n = " + str(3))  
    if n == 1:  
        return 1  
    else:  
        Sum = 3**2 + Summation(3-1)  
        return Sum
```

```
def Summation(2):  
    print("Calling summation with n = " + str(2))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(2-1)  
        return Sum
```

```
def Summation(1):  
    print("Calling summation with n = " + str(1))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(1-1)  
        return Sum
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
print(Summation(4))
```

```
def Summation(3):  
    print("Calling summation with n = " + str(3))  
    if n == 1:  
        return 1  
    else:  
        Sum = 3**2 + Summation(3-1)  
        return Sum
```

```
def Summation(2):  
    print("Calling summation with n = " + str(2))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(2-1)  
        return Sum
```

```
def Summation(1):  
    print("Calling summation with n = " + str(1))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(1-1)  
        return Sum
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
print(Summation(4))
```

```
def Summation(3):  
    print("Calling summation with n = " + str(3))  
    if n == 1:  
        return 1  
    else:  
        Sum = 3**2 + Summation(3-1)  
        return Sum
```

```
def Summation(2):  
    print("Calling summation with n = " + str(2))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(2-1)  
        return Sum
```

```
def Summation(1):  
    print("Calling summation with n = " + str(1))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(1-1)  
        return Sum
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
print(Summation(4))
```

```
def Summation(3):  
    print("Calling summation with n = " + str(3))  
    if n == 1:  
        return 1  
    else:  
        Sum = 3**2 + Summation(3-1)  
        return Sum
```

```
def Summation(2):  
    print("Calling summation with n = " + str(2))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(2-1)  
        return Sum
```

```
def Summation(1):  
    print("Calling summation with n = " + str(1))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(1-1)  
        return Sum
```

```
def Summation(4):  
    print("Calling summation with n = " + str(4))  
    if n == 1:  
        return 1  
    else:  
        Sum = 4**2 + Summation(4-1)  
        return Sum  
print(Summation(4))
```

```
def Summation(3):  
    print("Calling summation with n = " + str(3))  
    if n == 1:  
        return 1  
    else:  
        Sum = 3**2 + Summation(3-1)  
        return Sum
```

```
def Summation(2):  
    print("Calling summation with n = " + str(2))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(2-1)  
        return Sum
```

```
def Summation(1):  
    print("Calling summation with n = " + str(1))  
    if n == 1:  
        return 1  
    else:  
        Sum = 2**2 + Summation(1-1)  
        return Sum
```

```
def Summation(4):
    print("Calling summation with n = " + str(4))
    if n == 1:
        return 1
    else:
        Sum = 4**2 + Summation(4-1)
        return Sum
print(Summation(4))
```

```
def Summation(3):
    print("Calling summation with n = " + str(3))
    if n == 1:
        return 1
    else:
        Sum = 3**2 + Summation(3-1)
        return Sum
```

```
def Summation(2):
    print("Calling summation with n = " + str(2))
    if n == 1:
        return 1
    else:
        Sum = 2**2 + Summation(2-1)
        return Sum
```

```
def Summation(1):
    print("Calling summation with n = " + str(1))
    if n == 1:
        return 1
    else:
        Sum = 2**2 + Summation(1-1)
        return Sum
```

```
def Summation(4):
    print("Calling summation with n = " + str(4))
    if n == 1:
        return 1
    else:
        Sum = 4**2 + Summation(4-1)
        return Sum
print(Summation(4))
```

```
def Summation(3):
    print("Calling summation with n = " + str(3))
    if n == 1:
        return 1
    else:
        Sum = 3**2 + Summation(3-1)
        return Sum
```

```
def Summation(2):
    print("Calling summation with n = " + str(2))
    if n == 1:
        return 1
    else:
        Sum = 2**2 + Summation(2-1)
        return Sum
```

```
def Summation(1):
    print("Calling summation with n = " + str(1))
    if n == 1:
        return 1
    else:
        Sum = 2**2 + Summation(1-1)
        return Sum
```

```
def Summation(4):
    print("Calling summation with n = " + str(4))
    if n == 1:
        return 1
    else:
        Sum = 4**2 + Summation(4-1)
        return Sum
print(Summation(4))
```

```
def Summation(3):
    print("Calling summation with n = " + str(3))
    if n == 1:
        return 1
    else:
        Sum = 3**2 + Summation(3-1)
        return Sum
```

```
def Summation(2):
    print("Calling summation with n = " + str(2))
    if n == 1:
        return 1
    else:
        Sum = 2**2 + Summation(2-1)
        return Sum
```

```
def Summation(1):
    print("Calling summation with n = " + str(1))
    if n == 1:
        return 1
    else:
        Sum = 2**2 + Summation(1-1)
        return Sum
```

Recursion

```
def Summation(n):
    print("Calling summation with n = " + str(n))
    if n == 1:
        return 1
    else:
        Sum = n**2 + Summation(n-1)
        print("Intermediate value for {}**2 + Summation({})
              is: {}".format(n, n-1, Sum))
        return Sum
print(Summation(3))
```

```
Calling summation with n = 3
Calling summation with n = 2
Calling summation with n = 1
Intermediate value for 2**2 + Summation(1) is: 5
Intermediate value for 3**2 + Summation(2) is: 14
14
```


Recursion

```
def FibNumber(n):  
    if n == 0:  
        return 0  
    elif n == 1:  
        return 1  
    else:  
        return FibNumber(n-1) + FibNumber(n-2)  
print("The 5th fibonacci number is = " , FibNumber(5))
```

```
Fib = 5
```

Recursion

```
DicFib = {0:0, 1:1}  
  
def FibNumber(n):  
    if not n in DicFib:  
        DicFib[n] = FibNumber(n-1) + FibNumber(n-2)  
        print(DicFib[n])  
    return DicFib[n]  
  
print("Fib = " , FibNumber(5))
```

```
1  
2  
3  
5  
Fib = 5
```