Break , Continue

The break statement - used to terminate the execution of the nearest enclosing loop, when the compiler encounters the break the control passes to the statement that follows the loop.

Example - for i in range(1,10): if i==5: print("i = ",i) break print("Outside break")

Output - Outside break Outside break Outside break Outside break i = 5

In [3]:

```
for i in range(1,10):
                if i==5:
                    print("i = ",i)
                    break
#                 print("Outside break")
print("Outside Loop")
```

```
i =  5
Outside Loop
```

Continue - Like break statement continue appear only inside loop when compiler encounters continue it skips all the statement in a loop. And the control tansfers to loop continuation portation of the nearest enclosing loop.

Example - for i in range(1,20): if i >=11 and i<=14: continue print(i) print("Done")

In [13]:

```
for i in range(1,20):
           if i >=11 and i<=14:
                continue
           print(i,end=" ")
print("Done")
```

```
1 2 3 4 5 6 7 8 9 10 15 16 17 18 19 Done
```

Pass statement. The Pass statement is used when a statement is required syntactically but no command or code has to be executed. It is also known as no operation statement.

Example - for letter in "Hello": pass print("pass",letter) print("Done")

Pass is executable statement where as comment is ignored by the compiler.

If we have a loop that is not implemented but we wish to extend it in future in such cases we can use pass, because a loop cannot have empty body.

In [18]:

```
for letter in "Hello":
            pass # The statement is doing nothing.
            print("pass",letter)
print("Done")
```

```
Done
```

Difference between Comment VS Pass

1. Pass is a null statement where as comment is ignored by the interpreter.

2. Pass is executable statement where as comment is ignored by the compiler.

Function

When we write a function in Python def is used with the name of the function and parameter list ended with the colon. By default every function returns a value, if explicitly not mentioned 'none' is returned.

In [24]:

```
# keyword, Function Name, Parameter(x,y):
def diff(x,y):
    return (x-y)

x = 10
y = 20

# renaming fuction :
# function name can be assigned to a variable

operation = diff
print(operation(x,y))
print(type(operation))
```

```
None
<class 'function'>
```

In [29]:

```
# WAP to call a built-in function 10 times within a function.

def call():
    for i in range(1,11):
        print(" Hello World ",i)

call()
```

```
 Hello World  1
 Hello World  2
 Hello World  3
 Hello World  4
 Hello World  5
 Hello World  6
 Hello World  7
 Hello World  8
 Hello World  9
 Hello World  10
```

Specification about Parameters.

1. The function name and number of arguments in function call must be same as that given in function definition.
2. If there is mismatch within the number of arguments and parameters passed the error will occur.
3. Arguments may be passed in the form of expression to the called function in such cases arguments are first evalueted and converted to the type of the formal parameter and then the function body gets executed.

In [31]:

```
# Example -
def func (i):
    print(i)
func((5+2*3)/9)
# We dont need to define the return datatype value
```

```
1.2222222222222223
```

Local Variable , Scope & Lifetime

Scope - Part of a program in which variable is accessable. Lifetime - The duration for which the variable exist.

In [32]:

```
var = "Good"
def show():
    global var1
    var1 = "Morning"
    print("In function var is: ",var)

show()
print("Outside function var1 is: ",var1)
print("var is: ",var)
```

```
In function var is:  Good
Outside function var1 is:  Morning
var is:  Good
```

We can have a variable with the same name as that of a global variable. In such cases new local variable is different from the global variable.

In [39]:

```
var = "Good"

def show():
    var = "Morning"
    print(var,end=" ")

show()
print(var,end=" ")
```

```
Morning Good
```

In [36]:

```
var = "Good"

def show():
    global var
    var = "Morning"
    print(var,end=" ")

show()
print(var,end=" ")
```

```
Morning Morning
```

Nested Function

In [40]:

```
def outer_func():
    outer_var = 10
    def inner_func():
        inner_var = 20
        print(inner_var)
        print(outer_var)
    inner_func()
    print("Outer Variable: ",outer_var)
    print("Inner Variable: ",inner_var)
outer_func()

# Inner variable cannot be accessed.
```

```
20
10
Outer Variable:  10

---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_11912\1034335432.py in <module>
      8     print("Outer Variable: ",outer_var)
      9     print("Inner Variable: ",inner_var)
---> 10 outer_func()

~\AppData\Local\Temp\ipykernel_11912\1034335432.py in outer_func()
      7     inner_func() # 20 10
      8     print("Outer Variable: ",outer_var)
----> 9     print("Inner Variable: ",inner_var)
     10 outer_func()

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

Return Statement

Every function has a implicit return statement as a last instruction in the function body. The implicit return statement returns nothing to its caller, so it is said to return 'none'.

In [44]:

```
def display(str):
    print(str)

x = display("Hello World")

print(x)
print(display("Hello Again"))
```

```
Hello World
None
Hello Again
None
```

In [ ]:

```
```