break statement

The break statement enables a program to skip over a part of the code. A break statement terminates the very loop it lies within.

example

```
for i in range(1,101,1):
   print(i ,end=" ")
   if(i==50):
      break
   else:
      print("Mississippi")
print("Thank you")
```

output

```
1 Mississippi
2 Mississippi
3 Mississippi
4 Mississippi
5 Mississippi
.
.
.
50 Mississippi
```

Continue Statement

The continue statement skips the rest of the loop statements and causes the next iteration to occur.

example

```
for i in [2,3,4,6,8,0]:
   if (i%2!=0):
      continue
   print(i)
```

output

```
2
4
6
8
0
```

Python Functions

A function is a block of code that performs a specific task whenever it is called. In bigger programs, where we have large amounts of code, it is advisable to create or use existing functions that make the program flow organized and neat.

There are two types of functions:

1. Built-in functions
2. User-defined functions

Built-in functions:

These functions are defined and pre-coded in python. Some examples of built-in functions are as follows:

min(), max(), len(), sum(), type(), range(), dict(), list(), tuple(), set(), print(), etc.

User-defined functions:

We can create functions to perform specific tasks as per our needs. Such functions are called user-defined functions.

Syntax:

```
def function_name(parameters):
  pass
  # Code and Statements
```

- Create a function using the def keyword, followed by a function name, followed by a paranthesis (()) and a colon(:).
- Any parameters and arguments should be placed within the parentheses.
- Rules to naming function are similar to that of naming variables.
- Any statements and other code within the function should be indented.

Calling a function:

We call a function by giving the function name, followed by parameters (if any) in the parenthesis.

Example:

```
def name(fname, lname):
   print("Hello,", fname, lname)
```

```
name("Sam", "Wilson")
```
Output:

```
Hello, Sam Wilson
```

Function Arguments and return statement

There are four types of arguments that we can provide in a function:

- Default Arguments
- Keyword Arguments
- Variable length Arguments
- Required Arguments

Default arguments:

We can provide a default value while creating a function. This way the function assumes a default value even if a value is not provided in the function call for that argument.

Example:

```
def name(fname, mname = "Jhon", lname = "Whatson"):
    print("Hello,", fname, mname, lname)

name("Amy")
```
Output:

```
Hello, Amy Jhon Whatson
```

Keyword arguments:

We can provide arguments with key = value, this way the interpreter recognizes the arguments by the parameter name. Hence, the the order in which the arguments are passed does not matter.

Example:

```
def name(fname, mname, lname):
   print("Hello,", fname, mname, lname)

name(mname = "Peter", lname = "Wesker", fname = "Jade")
```

Output:

```
Hello, Jade Peter Wesker
```

Required arguments:

In case we don’t pass the arguments with a key = value syntax, then it is necessary to pass the arguments in the correct positional order and the number of arguments passed should match with actual function definition.

Example 1: when number of arguments passed does not match to the actual function definition.

```
def name(fname, mname, lname):
   print("Hello,", fname, mname, lname)

name("Peter", "Quill")
```

Output:

```
name("Peter", "Quill")\
TypeError: name() missing 1 required positional argument: 'lname'
```

Example 2: when number of arguments passed matches to the actual function definition.

```
def name(fname, mname, lname):
   print("Hello,", fname, mname, lname)

name("Peter", "Ego", "Quill")
```

Output:

```
Hello, Peter Ego Quill
```

Variable-length arguments:

Sometimes we may need to pass more arguments than those defined in the actual function. This can be done using variable-length arguments.

There are two ways to achieve this:

Arbitrary Arguments:
While creating a function, pass a * before the parameter name while defining the function. The function accesses the arguments by processing them in the form of tuple.

Example:

```
def name(*name):
   print("Hello,", name[0], name[1], name[2])

name("James", "Buchanan", "Barnes")
```

Output:

```
Hello, James Buchanan Barnes
```

Keyword Arbitrary Arguments:

While creating a function, pass a * before the parameter name while defining the function. The function accesses the arguments by processing them in the form of dictionary.

Example:

```
def name(**name):
   print("Hello,", name["fname"], name["mname"], name["lname"])

name(mname = "Buchanan", lname = "Barnes", fname = "James")
```

Output:

```
Hello, James Buchanan Barnes
```

return Statement

The return statement is used to return the value of the expression back to the calling function.

Example:

```
def name(fname, mname, lname):
   return "Hello, " + fname + " " + mname + " " + lname

print(name("James", "Buchanan", "Barnes"))
```

Output:

```
Hello, James Buchanan Barnes
```