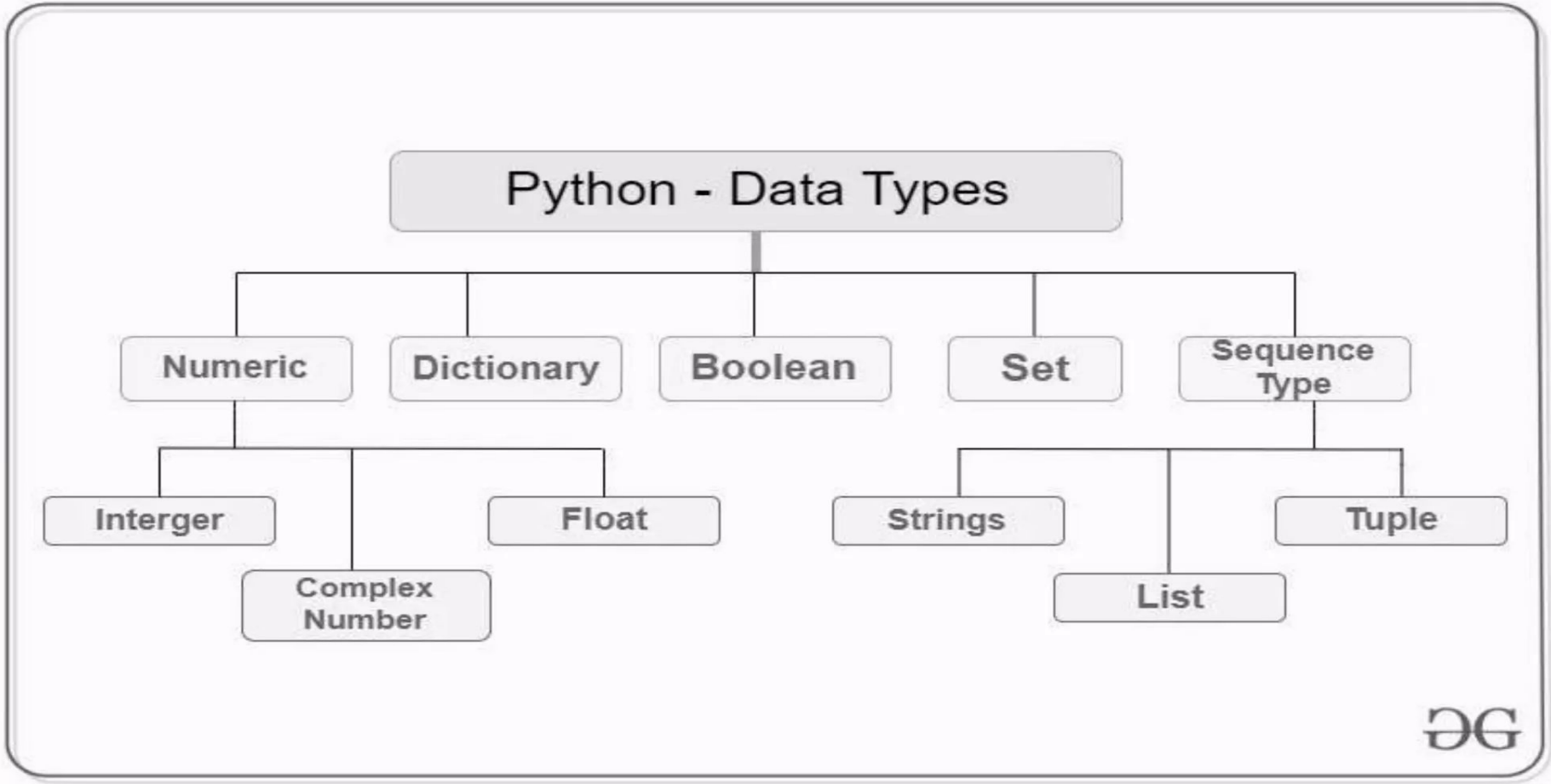
Programming with Python

Department of CE-AI/ CE-Big data

Unit no. 2 Python Data Types and Program Flow Controls


Python - Data Types
Numeric
Dictionary
Boolean
Set
Sequence Type
Interger
Complex Number
Float
Strings
List
Tuple

Numeric

The numeric data type in Python represents the data that has a numeric value. A numeric value can be an integer, a floating number, or even a complex number. These values are defined as: 1. integer

2. float, and

3. complex

- **Integers** – This value is represented by int class. It contains positive or negative whole numbers (without fractions or decimals). In Python, there is no limit to how long an integer value can be.

For Example: x=10

- **Float** – This value is represented by the float class. It is a real number with a floating-point representation. It is specified by a decimal point.

For Example: x=10.0

- **Complex Numbers** – A complex number is represented by a complex class. It is specified as *(real part) + (imaginary part)j*.

For example – 2+3j.

Example	Data Type
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex

Sequence data types

Sequences allow you to store multiple values in an organized and efficient manner.

1. List
2. Tuple
3. String

List

- List is an heterogeneous collection of items of various data types.
- Lists are used to store multiple items in a single variable.
- The list is mutable or changeable, meaning that we can change, add, and remove items in a list after it has been created.
- List is created by using [].

For Example:

```
list=[1,2,"hello",12.0]
```

List Methods or Built-in functions

Python has a set of built-in methods that you can use on lists.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list

count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value

reverse()	Reverses the order of the list
sort()	Sorts the list

Tuple

- A tuple is also a heterogeneous collection of python objects separated by commas.
- Tuples are used to store multiple items in a single variable.
- Tuple is immutable or unchangeable.
- Tuple is created by using ().

For Example:

```
thistuple = ("apple", "banana", "cherry")
```

Tuple methods

Python has two built-in methods that you can use on tuples.

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

String

Strings in python are surrounded by either single quotation marks, or double quotation marks.

It is represented by str.

For example:

```
x="hello"
```

Slicing in strings

- You can return a range of characters by using the slice syntax.
- Specify the start index and the end index, separated by a colon, to return a part of the string.

start: The starting index of the slice (inclusive).

stop: The ending index of the slice (exclusive).

step: The step between each index in the slice.

Syntax :-

x[start:stop:step]

For Example:

Without step:

```
x="good morning"
print(x[2:5])
```

With step:

```
x="good morning"
print(x[2:5:2])
```

Negative Indexing

Use negative indexes to start the slice from the end of the string:

For Example:

```
x="good morning"

print(x[-5:-2])
```

Python - Modify Strings

Python has a set of built-in methods that you can use on strings.

1. **Upper Case:**

The `upper()` method returns the string in upper case:

For Example:

```
x=”good morning”

print(x.upper())
```

2. Lower Case

The lower() method returns the string in lower case:

For Example:

```
x=”good morning”

print(x.lower())
```

3. Remove Whitespace

Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

The strip() method removes any whitespace from the beginning or the end:

For Example:

```
x=" good morning "

print(x.strip())
```

4. Replace String

The replace() method replaces a string with another string:

For Example:

```
x=”good morning”

print(x.replace("g", "H"))
```

5. Split String

The split() method returns a list where the text between the specified separator becomes the list items.

For Example:

```
x="good, morning"
print(x.split(","))
```

Python - String Concatenation

String Concatenation

To concatenate, or combine, two strings you can use the + operator.

For Example:

```
a = "Good"

b = "Morning"

c = a + b

print(c)
```

Python - Format - Strings

1. String Format

As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:

For Example:

```
age = 36

txt = "My name is John, I am " + age

print(txt)
```

2. f-string

To specify a string as an f-string, simply put an f in front of the string literal, and add curly brackets {} as placeholders for variables and other operations.

For Example:

```
age = 36

txt = f"My name is John, I am {age}"

print(txt)
```

Escape characters

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

For example:

txt = "We are the so-called "Vikings" from the north."

txt = "We are the so-called \"Vikings\" from the north."

Set

- Sets are used to store multiple items in a single variable.
- A set is a collection which is *unordered*, *unchangeable**, and *unindexed*.
- Duplicates are not allowed.
- Once a set is created, you cannot change its items, but you can add new items.

For Example:-

```
s= {"apple", "banana", "cherry", "apple"}

print(s)
```

Set Methods

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
pop()	Removes an element from the set
remove()	Removes the specified element

Dictionary in python

- Dictionaries are used to store data values in key:value pairs.
- A dictionary is a collection which is ordered*, changeable and do not allow duplicates.
- Dictionaries are written with curly brackets, and have keys and values:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Booleans

- Booleans represent one of two values: True or False.
- In programming you often need to know if an expression is True or False.
- When you compare two values, the expression is evaluated and Python returns the Boolean answer:
- For Example:

```
print(10 > 9)

print(10 == 9)

print(10 < 9)
```

Operators

Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Arithmetic operations

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y

/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3

*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3

&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y

<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Bitwise Operators

Bitwise operators are used to compare (binary)

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
|	OR	Sets each bit to 1 if one of two bits is 1	x | y

^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y
~	NOT	Inverts all the bits	~x
<< <	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2
>>>>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x >> 2

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

Python if Statement

An if statement executes a block of code only when the specified condition is met.

Syntax

```
if condition:
    # body of if statement
```

Here, condition is a boolean expression, such as number > 5, that evaluates to either True or False.

- If condition evaluates to True, the body of the if statement is executed.
- If condition evaluates to False, the body of the if statement will be skipped from execution.

Condition is True

```
number = 10
if number > 0:
    # code

# code after if
```

Condition is False

```
number = -5
if number > 0:
    # code

# code after if
```

Example of IF statement:-

```
a = 33
b = 200
if b > a:
 print("b is greater than a")
```

In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

Indentation

Python uses indentation to define a block of code, such as the body of an if statement.

For example,

If statement, without indentation (will raise an error):

```
a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error
```

In this example `a` is equal to `b`, so the first condition is not true, but the `elif` condition is true, so we print to screen that "a and b are equal".

Python if...else Statement

An if statement can have an optional else clause. The else statement executes if the condition in the if statement evaluates to False.

Syntax

```
if condition:
    # body of if statement
else:
    # body of else statement
```

Here, if the `condition` inside the `if` statement evaluates to

- **True** - the body of `if` executes, and the body of `else` is skipped.
- **False** - the body of `else` executes, and the body of `if` is skipped

Condition is True

```
number = 10
if number > 0:
    # code

else:
    # code

# code after if
```

Condition is False

```
number = -5
if number > 0:
    # code

else:
    # code

# code after if
```

Example of else statement:

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

Python if…elif…else Statement

The if...else statement is used to execute a block of code among two alternatives.

However, if we need to make a choice between more than two alternatives, we use the if...elif...else statement.

Syntax:

```
if condition1:
    # code block 1

elif condition2:
    # code block 2

else:
    # code block 3
```

1st Condition is True

```
let number = 5
if number > 0 :
    # code

elif number < 0 :
    # code

else :
    # code


# code after if
```

2nd Condition is True

```
let number = -5
if number > 0 :
    # code

elif number < 0 :
    # code

else :
    # code


# code after if
```

All Conditions are False

```
let number = 0
if number > 0 :
    # code

elif number < 0 :
    # code

else :
    # code


# code after if
```

Example of if…elif…else Statement

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

Python Nested if Statements

It is possible to include an if statement inside another if statement.

For example,

```
if (condition1):

    # executes when condition is True

    if (condition2):

        # executes when condition is True
```

Outer if Condition is True

```
number = 5
if number >= 0:
    if number == 0:
        #code
    else:
        #code
else:
    #code
```

Outer if Condition is False

```
number = -5
if number >= 0:
    if number == 0:
        #code
    else:
        #code
else:
    #code
```

Example of nested if statements

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

Python Loops

Python has two primitive loop commands:

- `while` loops
- `for` loops

Python for Loop

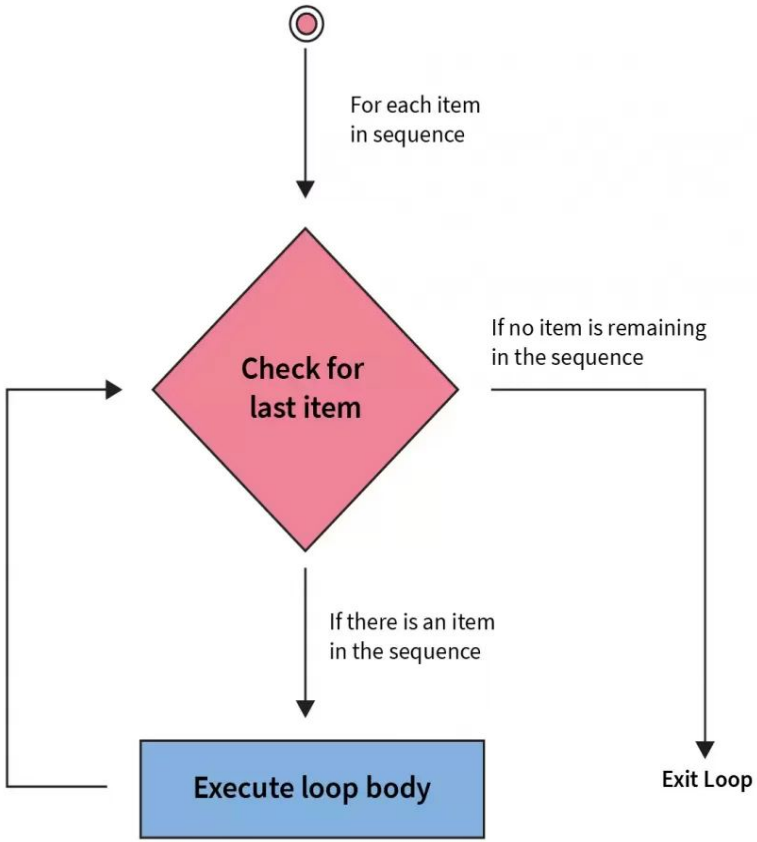
In Python, we use a for loop to iterate over sequences such as lists, strings, dictionaries, etc.

For example:

```
for val in sequence:
    # body of the loop
```

The for loop iterates over the elements of sequence in order. In each iteration, the body of the loop is executed.

The loop ends after the last item in the sequence is reached.


For each item
in sequence
Check for
last item
If no item is remaining
in the sequence
If there is an item
in the sequence
Execute loop body
Exit Loop

Example of for loop

```
languages = ['Swift', 'Python', 'Go']


# access elements of the list one by one
for lang in languages:
    print(lang)
```

for Loop with Python range()

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

For example:

```
for x in range(6):
  print(x)
```

Note that range(6) is not the values of 0 to 6, but the values 0 to 5.

The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

For example:

```
for x in range(2, 6):
  print(x)
```

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, 3):

```
for x in range(2, 30, 3):
  print(x)
```

Else in For Loop

The else keyword in a for loop specifies a block of code to be executed when the loop is finished:

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

Nested Loops

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]
for x in adj:
  for y in fruits:
    print(x, y)
```

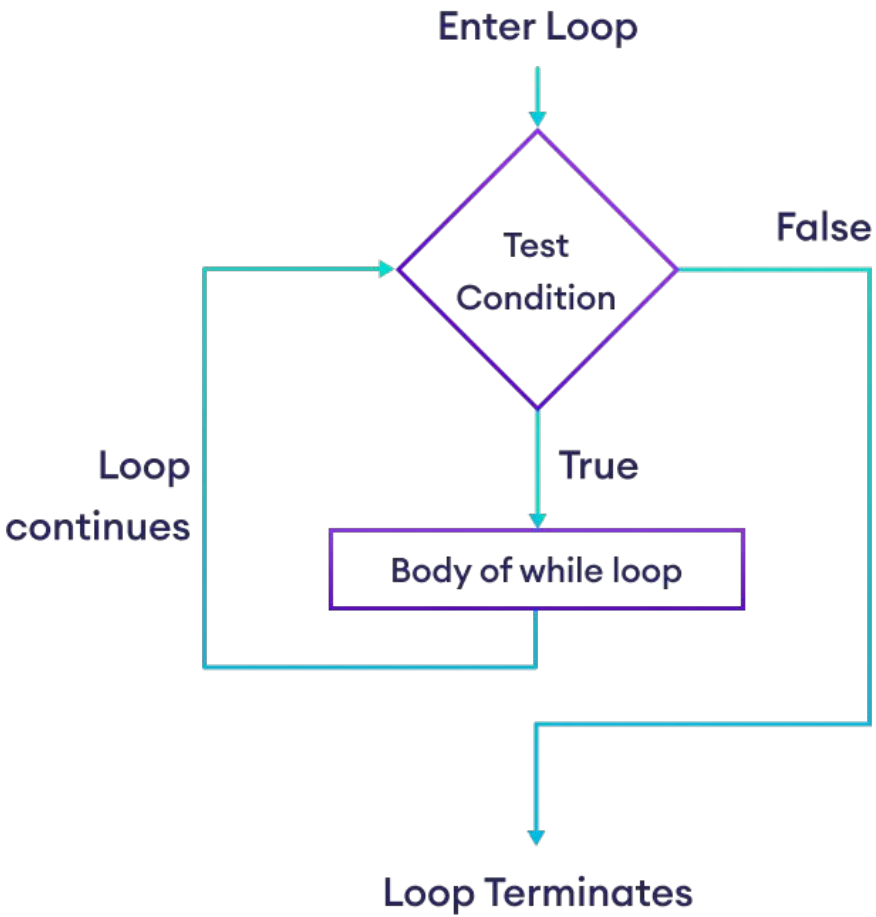
Python while Loop

In Python, we use a while loop to repeat a block of code until a certain condition is met. For example,

while Loop Syntax:

```
while condition:

 # body of while loop
```


Enter Loop
Test Condition
False
True
Loop continues
Body of while loop
Loop Terminates

For example

```
number = 1


while number <= 3:
    print(number)
     number = number + 1
```

Python while loop with an else clause

In Python, a `while` loop can have an optional `else` clause - that is executed once the loop condition is `False`. For example,

```
counter = 0
while counter  <  2:
    print('This is inside loop')
    counter = counter + 1
else:
     print('This is inside else block')
```

Python break and continue

In programming, the break and continue statements are used to alter the flow of loops:

- break exits the loop entirely
- continue skips the current iteration and proceeds to the next one

Python break Statement

The break statement terminates the loop immediately when it's encountered.

Syntax:-

```
break
```

Note: The break statement is usually used inside decision-making statements such as if...else.

```
for val in sequence:
    # code
    if condition:
        break

    # code
```

```
while condition:
    # code
    if condition:
        break

    # code
```

For Example:

```
for i in range(5):
    if i == 3:
        break
    print(i)
```

Python continue Statement

The continue statement skips the current iteration of the loop and the control flow of the program goes to the next iteration.

Syntax:-

```
continue
```

```
for val in sequence:
    # code
    if condition:
        continue

    # code
```

```
while condition:
    # code
    if condition:
        continue

    # code
```

For Example:

```
for i in range(5):
    if i == 3:
        continue
    print(i)
```

Python pass Statement

In Python programming, the pass statement is a null statement which can be used as a placeholder for future code.

Suppose we have a loop or a function that is not implemented yet, but we want to implement it in the future. In such cases, we can use the pass statement.

The syntax of the pass statement is:

```
pass
```

Example:

```
n = 10

# use pass inside if statement
if n > 10:
    pass

  print('Hello')
```