Python Notes

Python is a programming language that is easy to read and use. It is used in various industries such as web development, data science, and machine learning due to its simplicity and popularity. In this tutorial, we will introduce Python, explain the industries where it is used, discuss why it is popular, and highlight its advantages.

Basic Syntax in Python:

*#**This is a comment***
print("Hello, world!")

There are several built-in data types in Python, including:

- **Numbers:** In numbers we have integers, floating-point numbers, and complex numbers.
- **Strings:** A string is a sequence of characters that is enclosed in single or double quotes, it contains text.
- **Lists:** A list is a collection of items that are ordered and changeable.
- **Tuples:** A tuple is similar to a list, but it is immutable, which means that its values cannot be changed.
- **Sets:** A set is a collection of unique elements.
- **Dictionaries:** A dictionary is a collection of key-value pairs that are unordered and changeable.
- **Booleans:** A boolean is a data type that can only have two values: True or False.

Syntax/Code

```
mathmogul = 5 # integer
numeralninja = 2.5 # float
brainbooster = "AlmaBetter" # string
boolean = True # boolean
print(type(mathmogul))
print(type(numeralninja))
print(type(brainbooster))
print(type(boolean))
```

```
print(5 + 6) # addition
print(4 - 3) # subtraction
print(6 * 5) # multiplication
print(6 / 5) # division
print(7 % 5) # modulus
print(5 ** 3) # exponentiation
print(4 == 2) # equal to
print(4 != 2) # not equal to
print(6 > 4) # greater than
print(5 < 2) # less than
print(6 >= 6) # greater than or equal to
print(3 <= 5) # less than or equal to
print(3 > 2 and 8 < 10) # logical and
print(4 > 6 or 9 < 10) # logical or
print(not(3 > 4)) # logical not
```

Conditions

When used in a condition, the statement returns a Boolean result evaluating either True or False. When the specified value is found inside the sequence, the statement returns True. Whereas when it is not found, we get a False.

Else

The condition can be any expression that evaluates to a Boolean value (True or False). If the condition is True, the code block indented below the if statement will be executed. If the condition is False, the code block will be skipped.

Examples

```
num = 5
if num > 0:
    print("The number is positive.")
```

Else

The else statement allows you to execute a different block of code if the if condition is False.

Here's the basic syntax:

```
if condition:
    # code to execute if condition is true
else:
    # code to execute if condition is false
```

Elif

The elif statement allows you to check multiple conditions in sequence, and execute different code blocks depending on which condition is true.

Here's the basic syntax:

```
if condition1:
    # code to execute if condition1 is true
elif condition2:
    # code to execute if condition1 is false and condition2 is true
elif condition3:
    # code to execute if condition1 and condition2 are false, and condition3 is true
else:
    # code to execute if all conditions are false
```

Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string). This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

While Loop in Python

In python, a while loop is used to execute a block of statements repeatedly until a given condition is satisfied. And when the condition becomes false, the line immediately after the loop in the program is executed.

Syntax:

```
while expression:
    statement(s)
```

***example**:*

```
# Python program to illustrate while loop
count = 0
while (count < 3):
    count = count + 1
    print("Hello Geek")
```

output

```
Hello Geek
Hello Geek
Hello Geek
```

For Loop in Python

For loops are used for sequential traversal. For example: traversing a list or string or array etc. In Python, there is "for in" loop which is similar to for each loop in other languages. Let us learn how to use for in loop for sequential traversals.

Syntax:

```
for iterator_var in sequence:
    statements(s)
```

***example**:*

```
# Python program to illustrate
# Iterating over range 0 to n-1
n = 4
for i in range(0, n):
        print(i)
```

Output

```
0
1
2
3
```

Break, Continue and Pass in Python

BREAK

The break statement will cause the loop to end even if the while loop condition evaluates to True, or in the case of for loop. It will cause the control of the program to jump out of the **for** loop even if it satisfies the condition.

Example

```
nums = [7, 2, 3, 1, 5, 4, 6, 8, 9]
count = 0
while count < 7:
  print(nums[count])
  count += 1
  if nums[count] == 4:
    break
print("End")
```

```
OUTPUT
7
2
3
1
5
4
END
```

Pass Statement in Python

Let's now talk about doing nothing in Python. Seems odd? But yes, the pass statement does nothing. For doing nothing, it's a pretty useful statement. The syntax of the pass statement is the same as a break.
To do nothing inside a block of code, you can use the pass statement.

syntax

```
if 1 + 2 == 3:
  print("Correct math")
  pass
  print("This will also be printed.")
output
Correct math
This will also be printed.
```

Continue Statement in Python

Continue is also one of the useful loop control statements in python. It is almost the opposite of the break statement discussed above. Break terminates the loop, and the continue statement forces the program to execute the next iteration of the loop.

Example

```
for letter in 'python':
  if letter == 'o':
    continue
  print(letter)
```

Comprehensions in Python:

There are three types of comprehensions in Python:

- List comprehensions
- Set comprehensions
- Dictionary comprehensions

List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

The Syntax

```
newlist = [expression for item in iterable if condition == True]
```

Example

```
Only accept items that are not "apple":
newlist = [x for x in fruits if x != "apple"]
```

Set Comprehensions:

Set comprehensions are pretty similar to list comprehensions. The only difference between them is that set comprehensions use curly brackets { }. Let's look at the following example to understand set comprehensions.

Syntax

```
input_list = [1, 2, 3, 4, 4, 5, 6, 6, 6, 7, 7]
set_using_comp = {var for var in input_list if var % 2 == 0}
print("Output Set using set comprehensions:",
                    set_using_comp)
```

Dictionary Comprehensions:

Extending the idea of list comprehensions, we can also create a dictionary using dictionary comprehensions. The basic structure of a dictionary comprehension looks like below.

syntax

output_dict = {key:value for (key, value) in iterable if (key, value satisfy this condition)}

Defining function

def is the keyword for defining a function. The function name is followed by parameter(s) in (). The colon : signals the start of the function body, which is marked by indentation. Inside the function body, the return statement determines the value to be returned.

Syntax

ex1:

```
def my_function():
  print("Hello from a function")

my_function()
```

ex2:

```
def my_function(fname):
  print(fname + " Refsnes")
my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

Python Class Attributes

Class attributes are variables that are shared across all instances of a class. They represent characteristics of the class as a whole and are defined outside of any of the class methods.

class Student:

Syntax:

```
    num_students = 0  # class attribute
    def __init__(self, name, age):
        self.name = name
        self.age = age
        Student.num_students += 1  # access class attribute
# Create two student objects
student_1 = Student("John", 18)
student_2 = Student("Mary", 19)
# Print the total number of students
print(Student.num_students) # Output: 2
```