Object-oriented programming (OOP) (Python I) Lab 2

ASST.LEC Adnan Habeeb & Fatima Mohammed

Python Introduction

What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

web development (server-side),

software development,

mathematics,

system scripting.

What can Python do?

Python can be used on a server to create web applications.

Python can be used alongside software to create workflows.

Python can connect to database systems. It can also read and modify files.

Python can be used to handle big data and perform complex mathematics.

Python Introduction

Why Python?

Python works on different platforms (Windows, Mac, Linux, Pi, etc).

Python has a simple syntax similar to the English language.

Python has syntax that allows developers to write programs with fewer lines than some other programming languages.

Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.

Python Syntax

Python syntax can be executed by writing directly in the Command Line:

```
>>> print("Hello, World!")
Hello, World!
```

Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:

```
C:\Users\Your Name>python myfile.py
```

Python Syntax

Python Indentation

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

Example:

```
if 5 > 2:
  print("Five is greater than two!")
```

Output :

```
Five is greater than two!
```

Python will give you an error if you skip the indentation:

Example:

Syntax Error:

```
if 5 > 2:
print("Five is greater than two!")
```

Python Syntax

The number of spaces is up to you as a programmer, the most common use is four, but it has to be at least one.

Example:

```
if 5 > 2:
 print("Five is greater than two!")
if 5 > 2:
        print("Five is greater than two!")
```

Output :

```
 Five is greater than two!
 Five is greater than two!
```

You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

Example:

Syntax Error:

```
if 5 > 2:
 print("Five is greater than two!")
        print("Five is greater than two!")
```

Python Comments

Comments can be used to explain Python code.

Comments can be used to make the code more readable.

Comments can be used to prevent execution when testing code.

Creating a Comment

Comments starts with a #, and Python will ignore them:

Example:

```
#This is a comment

print("Hello, World!")
```

Output :

```
Hello, World!
```

A comment does not have to be text that explains the code, it can also be used to prevent Python from executing code:

Example:

```
#print("Hello, World!")

print("Cheers, Mate!")
```

Output :

```
Hello, World!
```

Python Comments

Multiline Comments

Python does not really have a syntax for multiline comments.

To add a multiline comment you could insert a # for each line:

Example:

```
#This is a comment

#written in

#more than just one line

print("Hello, World!")
```

Output :

```
Hello, World!
```

Python Comments

Multiline Comments

Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

Example:

```
"""

This is a comment

written in

more than just one line

"""

print("Hello, World!")
```

Output :

```
Hello, World!
```

Python Variables

Variables are containers for storing data values.

Creating Variables

Example:

```
x = 5

y = "John"

print(x)

print(y)
```

Output :

```
5
John
```

Get the Type

You can get the data type of a variable with the type() function.

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

Output :

```
<class 'int'>
<class 'str'>
```

Python Variables

Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

A variable name must start with a letter or the underscore character

A variable name cannot start with a number

A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)

Variable names are case-sensitive (age, Age and AGE are three different variables)

A variable name cannot be any of the Python keywords.

Legal variable names:

```
myvar = "John“       MYVAR = "John"

myvar2 = "John“      myVar = "John"

my_var = "John"

_my_var = "John"
```

Python Variables

Example :

Illegal variable names:

```
2myvar = "John"
my-var = "John"
my var = "John“
```

-Remember that variable names are case-sensitive

Many Values to Multiple Variables

Python allows you to assign values to multiple variables in one line:

Example Get your own Python Server :

```
x, y, z = "Orange", "Banana", "Cherry"

print(x)

print(y)

print(z)
```

Output :

```
Orange
Banana
Cherry
```

Python Data Types

In Python, the data type is set when you assign a value to a variable:

Example	Data Type
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = {"name" : "John", "age" : 36}	dict
x = {"apple", "banana", "cherry"}	set
x = True	bool
x = None	NoneType

Python Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + operator to add together two values:

Example:

```
print(10 + 5)
```

Output :

```
15
```

Python divides the operators in the following groups:

Arithmetic operators

Assignment operators

Comparison operators

Logical operators

Identity operators

Membership operators

Bitwise operators

Python Operators

Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical 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

Python Operators

Python 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

Python Operators

Python 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

Python Operators

Python 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)

Python Operators

Python 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

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

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the if , elif and else keyword.

Example

If 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")
```

Output :

```
a is greater than b
```

Nested If

You can have if statements inside if statements, this is called nested if statements.

Example

If statement:

```
x = 41

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

Output :

```
Above ten,
and also above 20!
```

if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

```
a = 33
b = 200

if b > a:
  pass
```

Python Loops

Python has two primitive loop commands:

- while loops
- for loops

The while Loop

With the while loop we can execute a set of statements as long as a condition is true.

```
i = 1
while i < 6:
  print(i)
  i += 1
```

Output :

```
1
2
3
4
5
```

Note: remember to increment i, or else the loop will continue forever.

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

The break Statement

With the break statement we can stop the loop even if the while condition is true:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

Output :

```
1
2
3
```

Python Loops

The continue Statement

With the continue statement we can stop the current iteration, and continue with the next:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

Output :

```
1
2
4
5
6
```

The else Statement

With the else statement we can run a block of code once when the condition no longer is true:

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

Output :

```
1
2
3
4
5
i is no longer less than 6
```

Python Loops

Python For 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).

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

Output :

```
apple
```

The continue Statement

With the continue statement we can stop the current iteration of the loop, and continue with the next:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

Output :

```
Apple
cherry
```

Python Loops

The range() Function

To loop through a set of code a specified number of times, we can use the range() function,

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

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

Output :

```
0 1 2 3 4 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 x in range(2, 6):
  print(x)
```

Output :

```
2 3 4 5
```

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)
```

Output :

```
2   5   8   11   14   17   20   23   26   29
```

Python Loops

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)
```

Output :

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

Python Lists

List

Lists are used to store multiple items in a single variable.
Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

Output :

```
["apple", "banana", "cherry"]
```

List Items

List items are ordered, changeable, and allow duplicate values.

List items are indexed, the first item has index [0], the second item has index [1] etc.

List Length

To determine how many items a list has, use the len() function:

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

Output :

```
3
```

Python Lists

```
thislist = []
print(thislist)
```

Empty list

```
thislist = [44.9, 54, 22, 2, 8, 22, “Python”]
```

0	1	2	3	4	5	6	**List Indexes**
-1	-2	-3	-4	-5	-6	-7	
44.9	54	22	2	8	33	Python	**List Items**

```
print(thislist[3])
print(thislist[-4])
```

Output :
2

```
print(thislist[-7])
print(thislist[6])
```

Output :
Python

```
print(thislist[6][3])
```

Output :
h

```
print(thislist[6][8])
```

Output :
Error out of range

```
print(thislist[0])
```

Output :
44.9

```
print(thislist[0][1])
```

Output :
Error (Float)

```
print(thislist[8])
```

Output :
Error out of range

Python Lists

Access Items

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[1])
print(thislist[-1])
print(thislist[2:5])
print(thislist[:4])
print(thislist[2:])
print(thislist[-4:-1])
```

Output :

```
banana
mango
['cherry', 'orange', 'kiwi']
['apple', 'banana', 'cherry', 'orange']
['cherry', 'orange', 'kiwi', 'melon',
'mango']
['orange', 'kiwi', 'melon']
```

Change Item Value

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

Output :

```
['apple', 'blackcurrant', 'cherry']
['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']
```

Python Lists

Insert Items

To insert a new list item, without replacing any of the existing values, we can use the insert() method.

The insert() method inserts an item at the specified index:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```

Output :

```
['apple', 'banana', 'watermelon', 'cherry']
```

Append Items

To add an item to the end of the list, use the append() method:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

Output :

```
['apple', 'banana', 'cherry', 'orange']
```

Extend List

To append elements from another list to the current list, use the extend() method.

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

Output :

```
['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya']
```

Python Lists

Remove Specified Item

The remove() method removes the specified item.

```
thislist = ["apple", "banana", "cherry", "banana", "kiwi"]
thislist.remove("banana")
print(thislist)
```

Output :

```
['apple', 'cherry', 'banana', 'kiwi']
```

Remove Specified Index

The pop() method removes the specified index.

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
thislist.pop()
print(thislist)
```

Output :

```
['apple']
```

Loop Through a List

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

Output :

```
apple
banana
cherry
```

Python Lists

Loop Through the Index Numbers

```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])
```

Output :

```
apple
banana
cherry
```

Sort Lists

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
```

Output :

```
['banana', 'kiwi', 'mango', 'orange', 'pineapple']
```

```
thislist = [100, 50, 65, 82, 23]
thislist.sort()
print(thislist)
```

Output :

```
[23, 50, 65, 82, 100]
```

Python Lists

Copy a List

You cannot copy a list simply by typing list2 = list1, because: list2 will only be a reference to list1, and changes made in list1 will automatically also be made in list2.

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

Output :

```
["apple", "banana", "cherry"]
```

Join Two Lists

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]
list3 = list1 + list2
print(list3)
```

Output :

```
['a', 'b', 'c', 1, 2, 3]
```

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]
for x in list2:
  list1.append(x)
print(list1)
```

Output :

```
['a', 'b', 'c', 1, 2, 3]
```

Python Lists

Reverse

Reverse() : Reverse the order of items in the list .

```
thislist = ["apple", "banana", "cherry"]
thislist.reverse()
print(thislist)
```

Output :

```
['cherry', 'banana', 'apple']
```

Count

Count(): Returns the count of the number of items passed as an argument

```
thislist = [2,4,7,3,5,3,5,2,2]
x=thislist.count(2)
print(x)
```

Output :

```
3
```

Index

Index() : Returns the index of the first matched item

```
thislist = [25,4,7,3,5,3,5,2,2]
x=thislist.index(2)
print(x)
```

Output :

```
7
```

Python Lists

Clear

clear() : Removes all items from the list.

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

Output :

```
[ ]
```

Thank you