Information and Communication Technology
Grade 13
Competency 9.6 and 9.7: Python programming

Reading Materials:

Introduction to Python

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

It is used for:

- web development (server-side),
- software development,
- mathematics,
- System scripting.

Python can be used for

- 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 can be used for rapid prototyping, or for production-ready software development.

Reasons for using Python

- Python works on different platforms (Windows, Mac, Linux, Raspberry 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 can be treated in a procedural way, an object-orientated way or a functional way.

Python Syntax compared to other programming languages

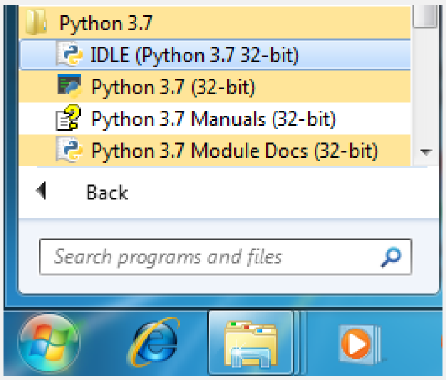
- Python was designed to for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

Python Install

Many PCs and Macs will have python already installed.

To check if you have python installed on a Windows PC, search in the start bar for Python or run the following on the Command Line (cmd.exe):

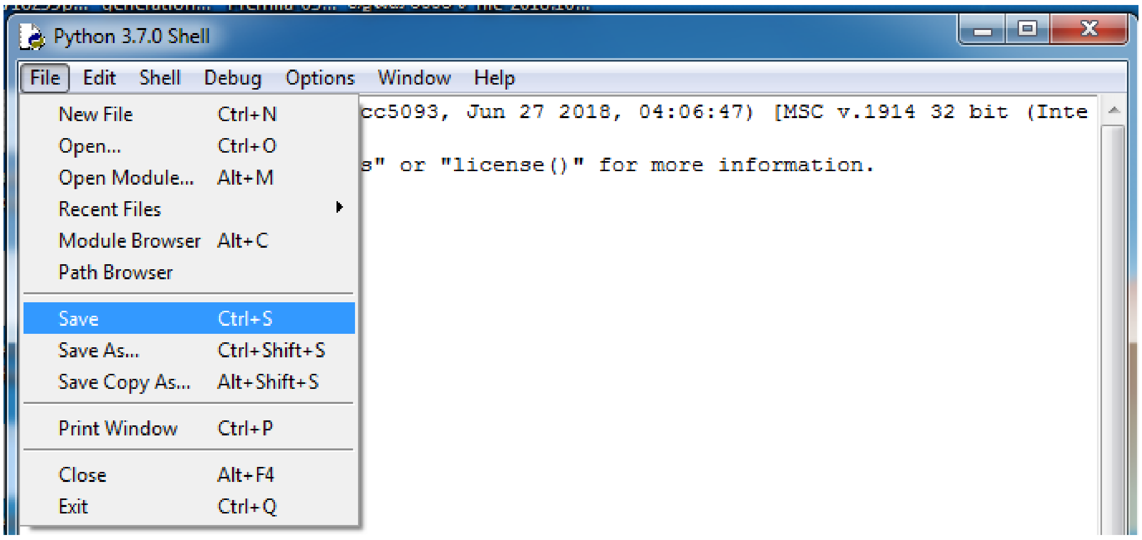
```
C:\Users\ Your Name>python –version
```



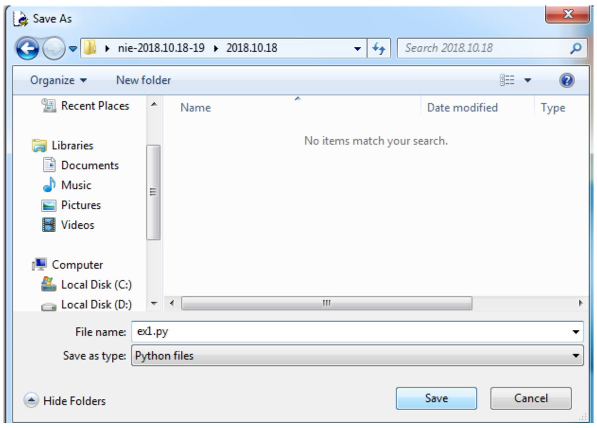
Python 3.7
IDLE (Python 3.7 32-bit)
Python 3.7 (32-bit)
Python 3.7 Manuals (32-bit)
Python 3.7 Module Docs (32-bit)
Back
Search programs and files

If you find that you do not have python installed on your computer, then you can download it for free from the following website: https://www.python.org/downloads/

Following on the IDLE (python 3.7):



```
cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte

s" or "license()" for more information.
```



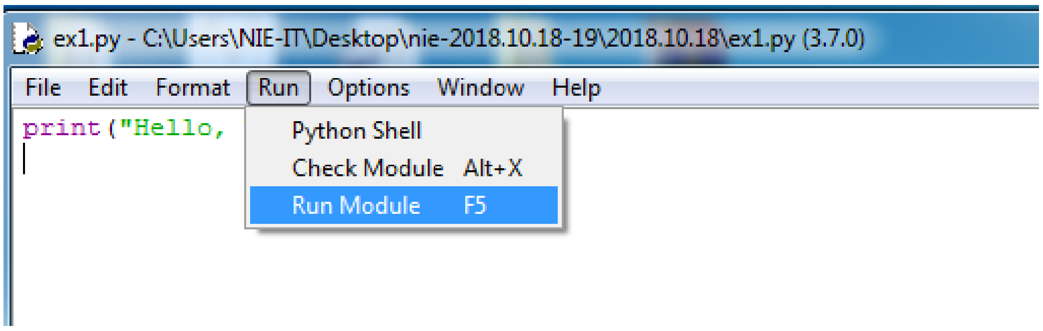
File name: ex1.py
Save as type: Python files

Where "ex1.py" is the name of your python file.

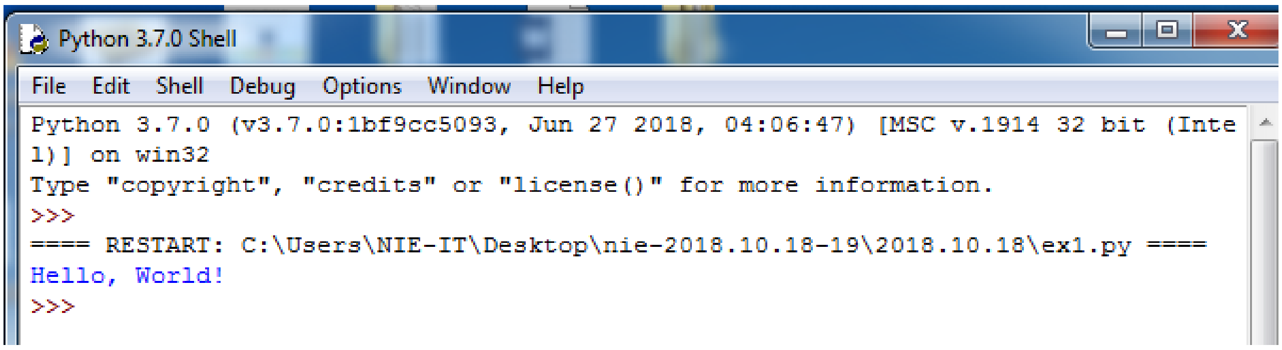
Let's write our first Python file, called ex1.py, which can be done in any text editor.

ex1.py

Write print("Hello, World!") and run that.

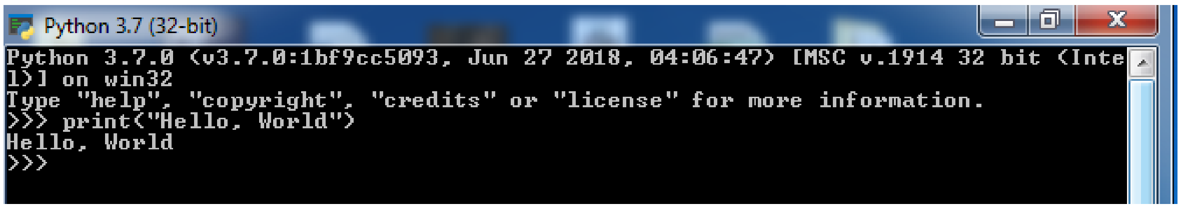

ex1.py - C:\Users\NIE-IT\Desktop\nie-2018.10.18-19\2018.10.18\ex1.py (3.7.0)
File Edit Format Run Options Window Help
Python Shell
Check Module Alt+X
Run Module F5

```
print("Hello,
```



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
==== RESTART: C:\Users\NIE-IT\Desktop\nie-2018.10.18-19\2018.10.18\ex1.py ====
Hello, World!
>>> 
```

Above programme in Command line

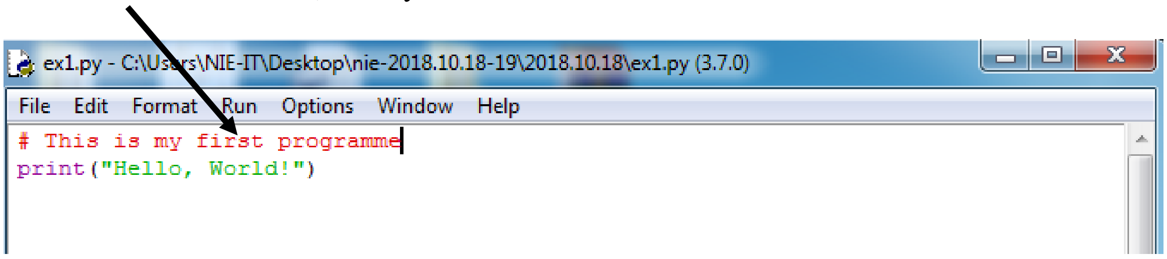


```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World")
Hello, World
>>>
```

Python Comments

Python has commenting capability for the purpose of in-code documentation.

Comments start with a #, and Python will render the rest of the line as a comment:


ex1.py - C:\Users\NIE-IT\Desktop\nie-2018.10.18-19\2018.10.18\ex1.py (3.7.0)
File Edit Format Run Options Window Help

```
# This is my first programme
print("Hello, World!")
```

```
# This is my first programme
print("Hello, World!")
```

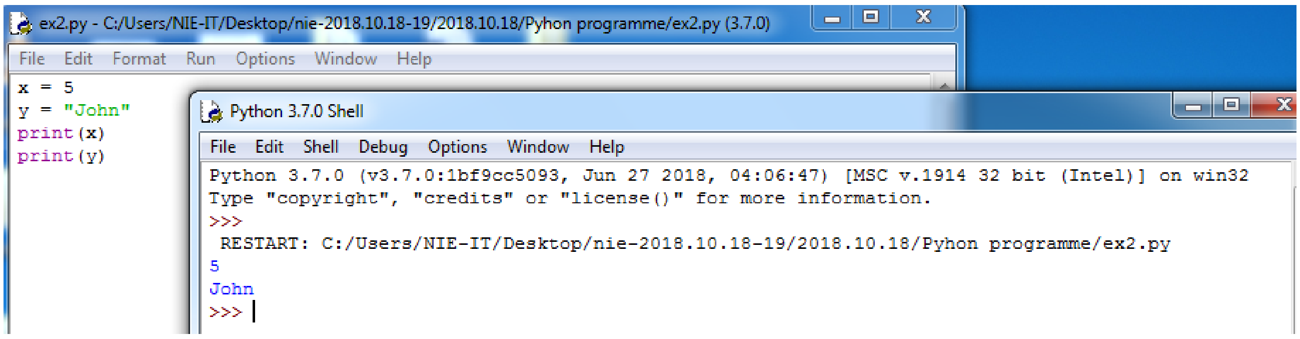
Python Variables

Creating Variables

Unlike other programming languages, Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

Example 2


ex2.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex2.py (3.7.0)
File Edit Format Run Options Window Help
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

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

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex2.py
5
John
>>> 
```

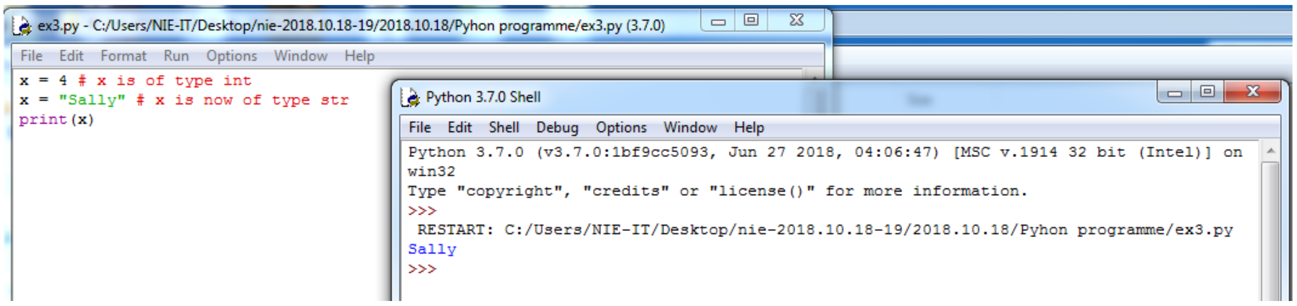
x = 5

y = "John"

print(x)

print(y)

Example 3


ex3.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex3.py (3.7.0)
File Edit Format Run Options Window Help
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
x = 4 # x is of type int
x = "Sally" # x is now of type str
print(x)
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex3.py
Sally
>>>
```

x = 4 # x is of type int

x = "Sally" # x is now of type str

print(x)

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:

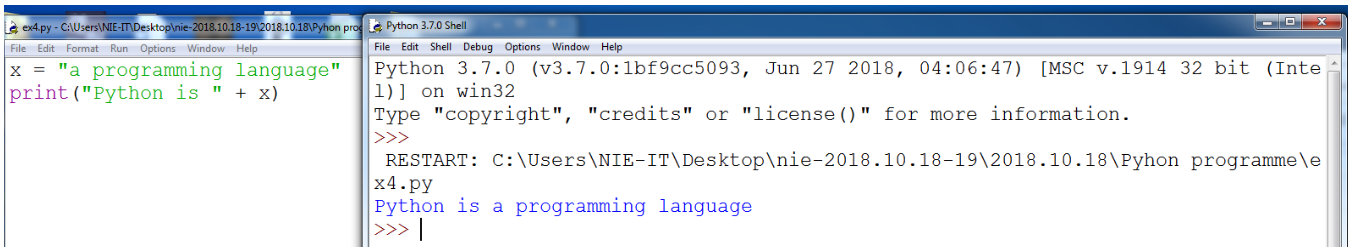
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Output Variables

The Python print statement is often used to output variables.

To combine both text and a variable, Python uses the `+` character:

Example 4



ex4.py - C:\Users\NIE-IT\Desktop\nie-2018.10.18-19\2018.10.18\Pyhon pro
File Edit Format Run Options Window Help
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

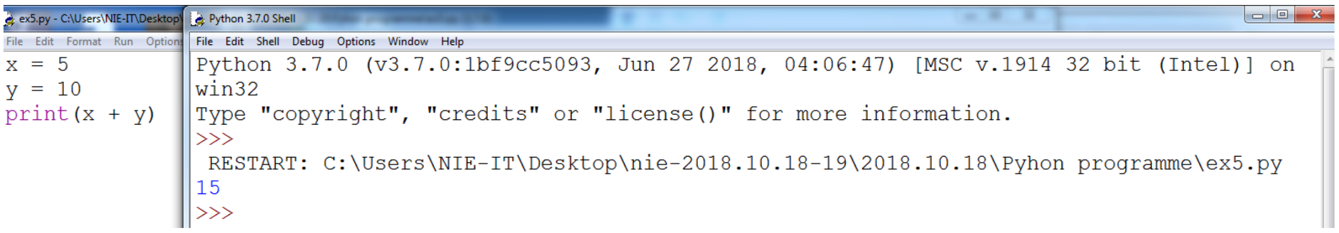
```
x = "a programming language"
print("Python is " + x)
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\NIE-IT\Desktop\nie-2018.10.18-19\2018.10.18\Pyhon programme\e
x4.py
Python is a programming language
>>> 
```

x = "a programming language"

print("Python is " + x)

Example 5



ex5.py - C:\Users\NIE-IT\Desktop\
File Edit Format Run Option
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
x = 5
y = 10
print(x + y)
```

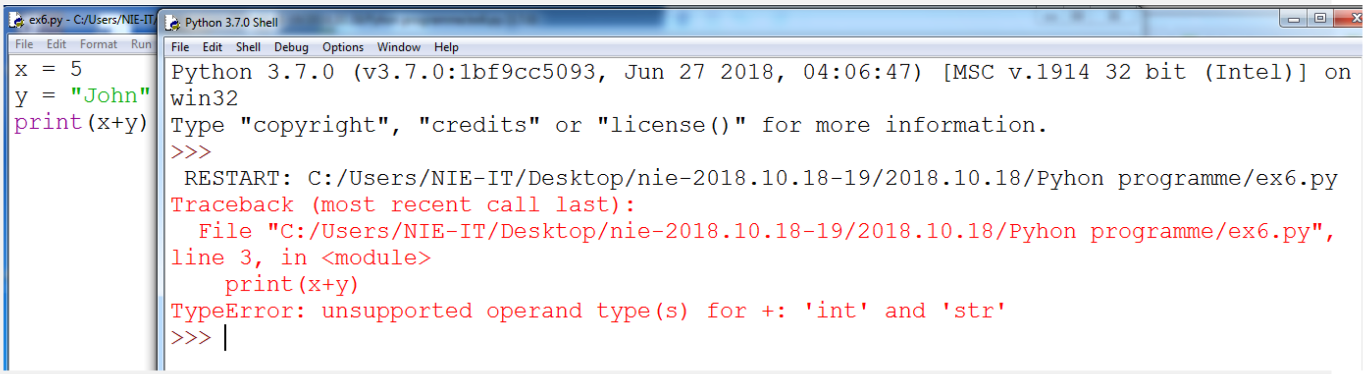
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\NIE-IT\Desktop\nie-2018.10.18-19\2018.10.18\Pyhon programme\ex5.py
15
>>>
```

x = 5

y = 10

print(x + y)

Example 6



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

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex6.py
Traceback (most recent call last):
  File "C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex6.py",
line 3, in <module>
    print(x+y)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> 
```

x = 5

y = "John"

print(x+y)

Python Numbers

There are three numeric types in Python:

- int
- float
- complex

Variables of numeric types are created when you assign a value to them:

```
x = 1   # int
y = 2.8 # float
z = 1j  # complex
```

Specify a Variable Type

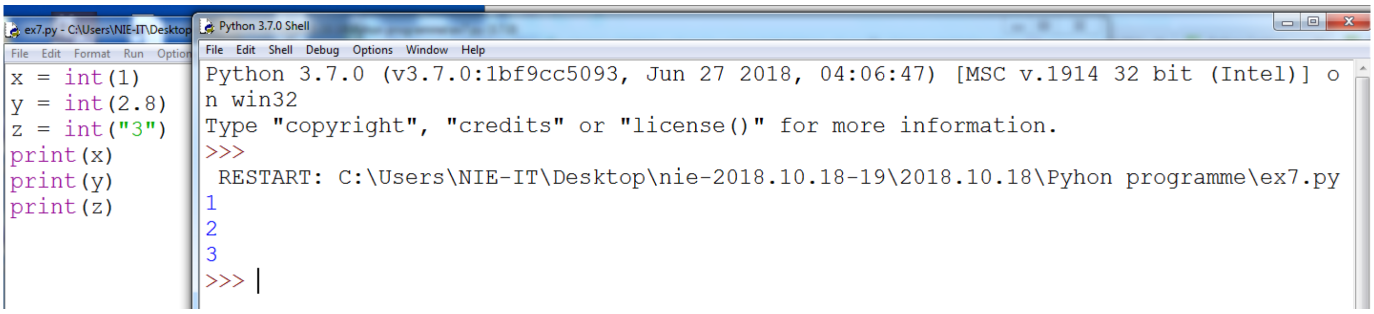
There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

- int() - constructs an integer number from an integer literal, a float literal (by rounding down to the previous whole number), or a string literal (providing the string represents a whole number)
- float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

Example 7

Integers:


ex7.py - C:\Users\NIE-IT\Desktop
File Edit Format Run Option
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
x = int(1)
y = int(2.8)
z = int("3")
print(x)
print(y)
print(z)
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] o
n win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\NIE-IT\Desktop\nie-2018.10.18-19\2018.10.18\Pyhon programme\ex7.py
1
2
3
>>> 
```

x = int(1)

y = int(2.8)

z = int("3")

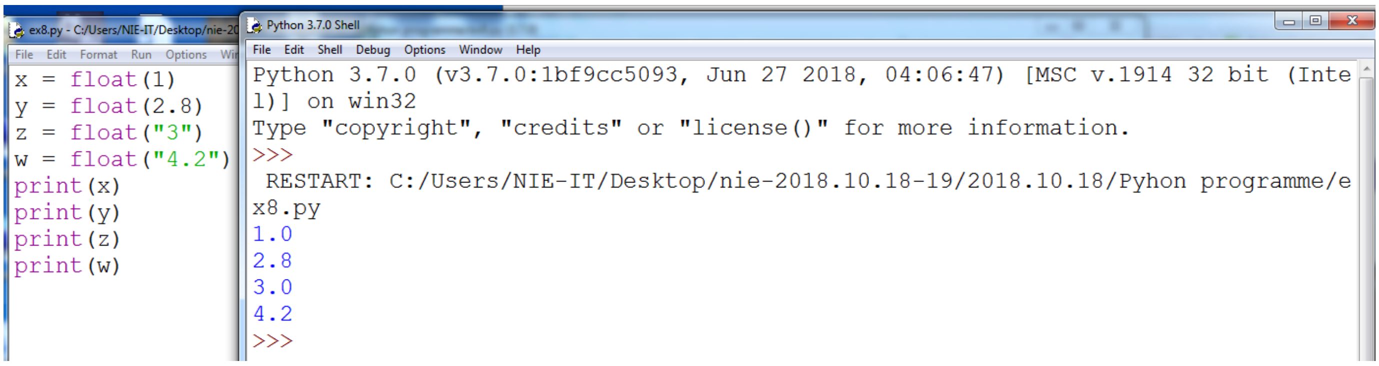
print(x)

print(y)

print(z)

Example 8

Floats:


ex8.py - C:/Users/NIE-IT/Desktop/nie-20
File Edit Format Run Options Wi
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
x = float(1)
y = float(2.8)
z = float("3")
w = float("4.2")
print(x)
print(y)
print(z)
print(w)
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/e
x8.py
1.0
2.8
3.0
4.2
>>>
```

x = float(1)

y = float(2.8)

z = float("3")

w = float("4.2")

print(x)

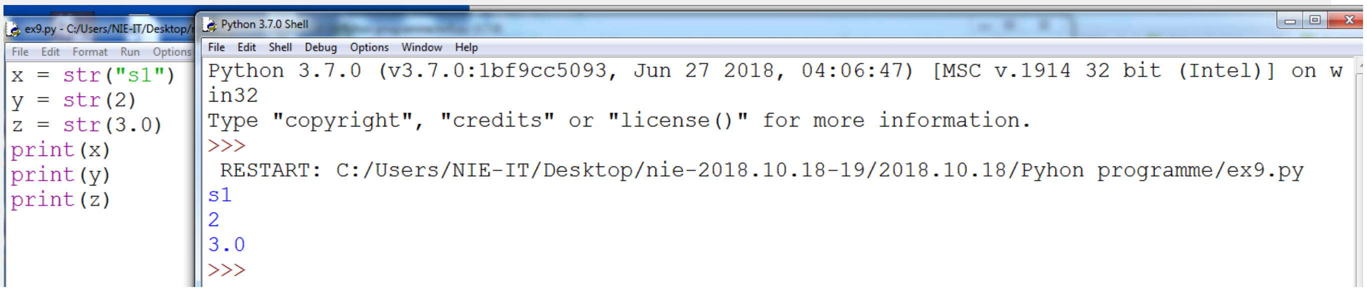
print(y)

print(z)

print(w)

Example 9

Strings:


ex9.py - C:/Users/NIE-IT/Desktop/
File Edit Format Run Options
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
x = str("s1")
y = str(2)
z = str(3.0)
print(x)
print(y)
print(z)
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on w
in32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex9.py
s1
2
3.0
>>>
```

x = str("s1")

y = str(2)

z = str(3.0)

print(x)

print(y)

print(z)

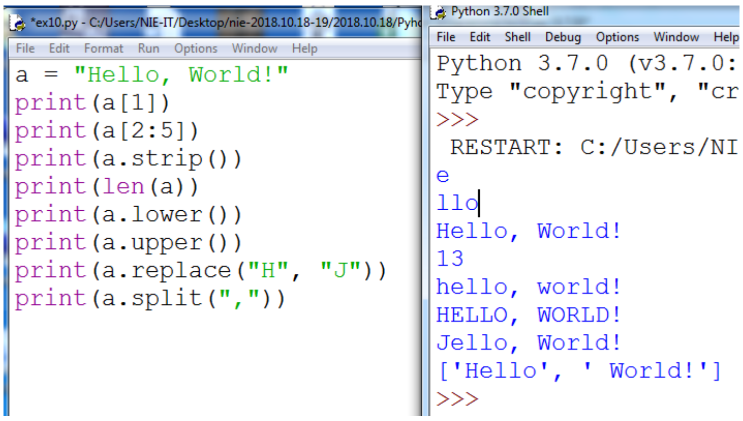
String Literals

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

'hello' is the same as "hello".

Strings can be output to screen using the print function. For example: print("hello").

Example 10


*ex10.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyho
File Edit Format Run Options Window Help
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
a = "Hello, World!"
print(a[1])
print(a[2:5])
print(a.strip())
print(len(a))
print(a.lower())
print(a.upper())
print(a.replace("H", "J"))
print(a.split(","))
```

```
Python 3.7.0 (v3.7.0:
Type "copyright", "cr
>>>
 RESTART: C:/Users/NI
e
llo
Hello, World!
13
hello, world!
HELLO, WORLD!
Jello, World!
['Hello', ' World!']
>>>
```

```
a = "Hello, World!"

print(a[1])

print(a[2:5])

print(a.strip())

print(len(a))

print(a.lower())

print(a.upper())

print(a.replace("H", "J"))

print(a.split(","))
```

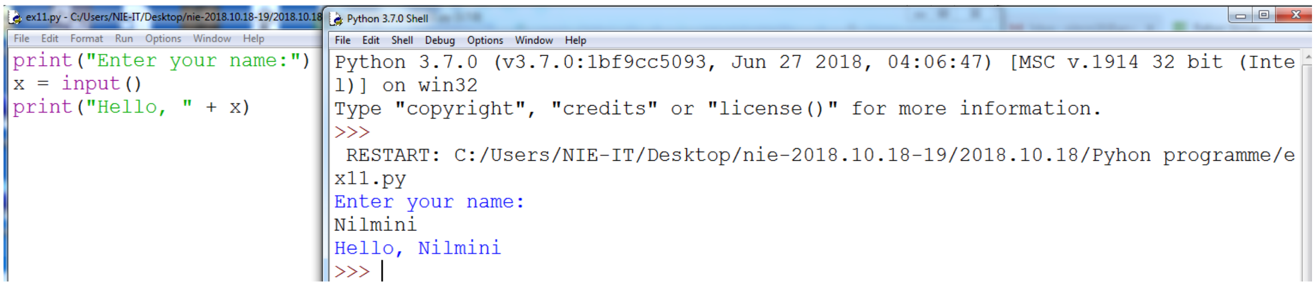
Command-line String Input

Python allows for command line input.

That means we are able to ask the user for input.

The following example asks for the user's name, then, by using the **input()** method, the program prints the name to the screen:

Example 11


ex11.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18
File Edit Format Run Options Window Help
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
print("Enter your name:")
x = input()
print("Hello, " + x)
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/e
x11.py
Enter your name:
Nilmini
Hello, Nilmini
>>> |
```

```
print("Enter your name:")

x = input()

print("Hello, " + x)
```

Python 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

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 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

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 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 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 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

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	(a & b) (means 0000 1100)
|	OR	Sets each bit to 1 if one of two bits is 1	(a | b) = 61 (means 0011 1101)
^	XOR	Sets each bit to 1 if only one of two bits is 1	(a ^ b) = 49 (means 0011 0001)
~	NOT	Inverts all the bits	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	a << 2 = 240 (means 1111 0000)
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	a >> 2 = 15 (means 0000 1111)

Python Operators Precedence

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift

&	Bitwise 'AND'td>
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Competency 9.8 and 9.9: Python programming

Control Structures - Intro, Selection

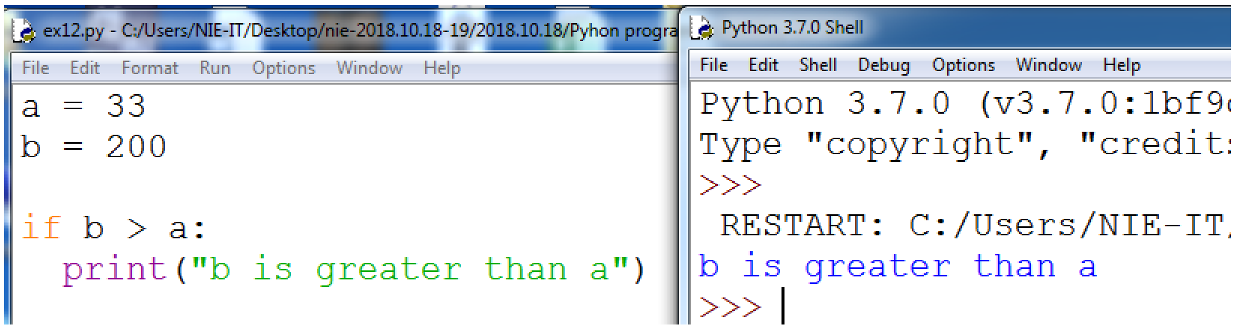
Flow of Control:

Flow of control through any given function is implemented with three basic types of control structures:

- **Sequential**: default mode. Sequential execution of code statements (one line after another) -- like following a recipe
- **Selection**: used for decisions, branching -- choosing between 2 or more alternative paths. In Python, these are the types of selection statements:
    - if
    - if-else
    - if-elif-else
- **Repetition**: used for looping, i.e. repeating a piece of code multiple times in a row. In Python, there are three types of loops:
    - while
    - for

Example 12

If statement:


ex12.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon progra
File Edit Format Run Options Window Help
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
a = 33
b = 200

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

```
Python 3.7.0 (v3.7.0:1bf9
Type "copyright", "credit
>>>
 RESTART: C:/Users/NIE-IT
b is greater than a
>>> 
```

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

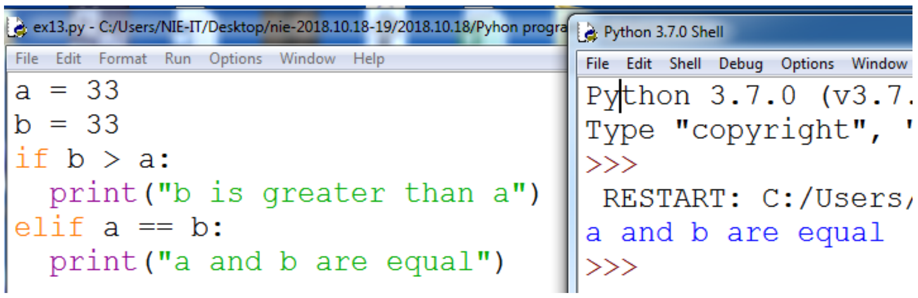
Indentation

Python relies on indentation, using whitespace, to define scope in the code. Other programming languages often use curly-brackets for this purpose.

Elif

The `elif` keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

Example 13

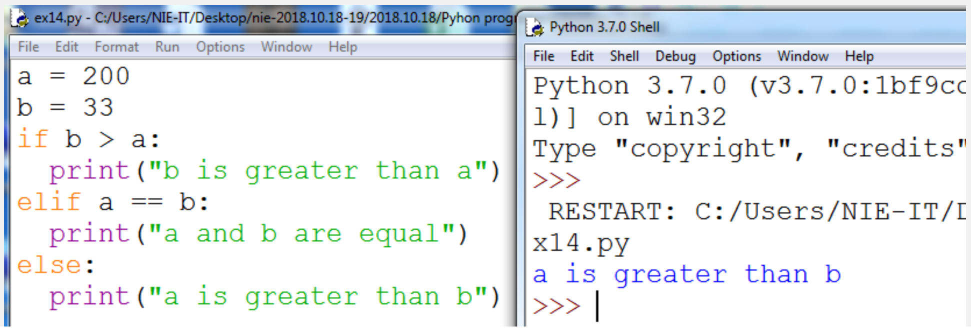

ex13.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon progra
File Edit Format Run Options Window Help
Python 3.7.0 Shell
File Edit Shell Debug Options Window

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

```
Python 3.7.0 (v3.7.
Type "copyright", '
>>>
 RESTART: C:/Users/
a and b are equal
>>>
```

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

Example 14


ex14.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon prog
File Edit Format Run Options Window Help
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
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 3.7.0 (v3.7.0:1bf9cc
1)] on win32
Type "copyright", "credits"
>>>
 RESTART: C:/Users/NIE-IT/D
x14.py
a is greater than b
>>>
```

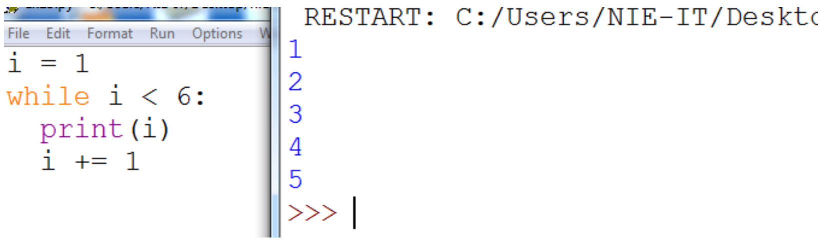
```
a = 200

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 Loops

Python has two primitive loop commands:

- while loops
- for loops

Example 15



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

```
RESTART: C:/Users/NIE-IT/Desktc
1
2
3
4
5
>>> 
```

i = 1

while i < 6:

  print(i)

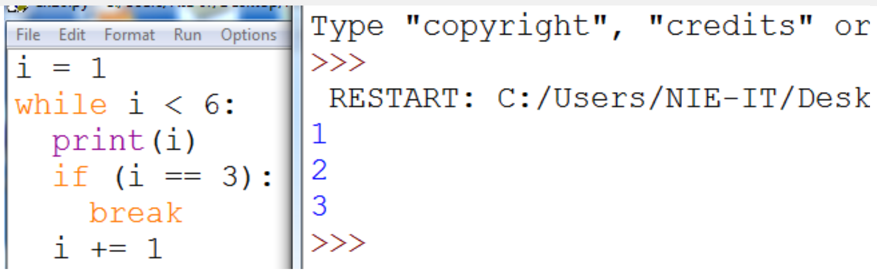
  i += 1

The break Statement

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

Example 16

Exit the loop when i is 3:



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

```
Type "copyright", "credits" or
>>>
 RESTART: C:/Users/NIE-IT/Desk
1
2
3
>>>
```

i = 1

while i < 6:

  print(i)

  if (i == 3):

    break

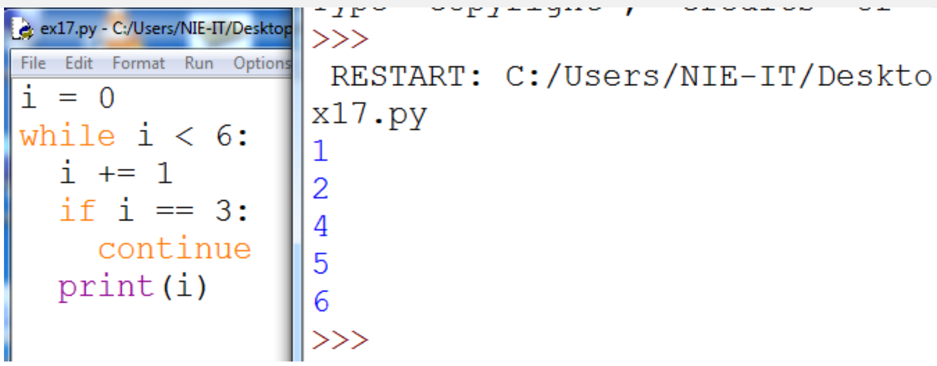
  i += 1

The continue Statement

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

Example 17

Continue to the next iteration if i is 3:



ex17.py - C:/Users/NIE-IT/Desktop

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

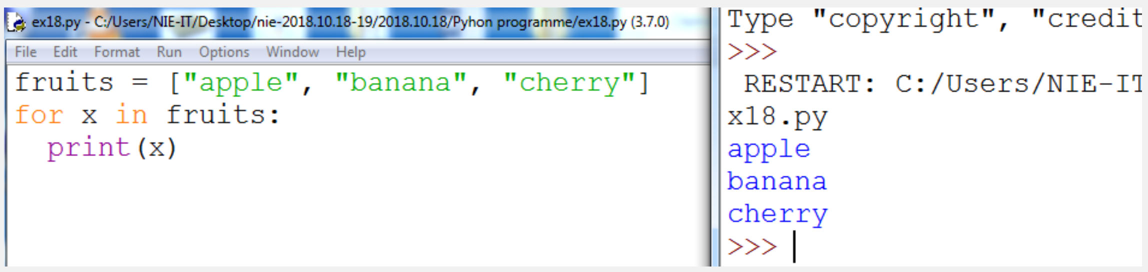
```
>>>
 RESTART: C:/Users/NIE-IT/Deskto
x17.py
1
2
4
5
6
>>>
```

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

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).

Example 18



ex18.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex18.py (3.7.0)

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

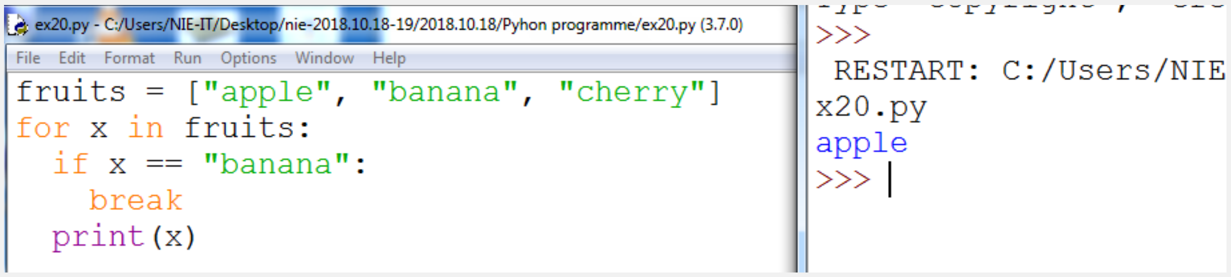
```
Type "copyright", "credit
>>>
 RESTART: C:/Users/NIE-IT
x18.py
apple
banana
cherry
>>> |
```

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

The break Statement

With the break statement we can stop the loop before it has looped through all the items:

Example 19



```
ex20.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex20.py (3.7.0)
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

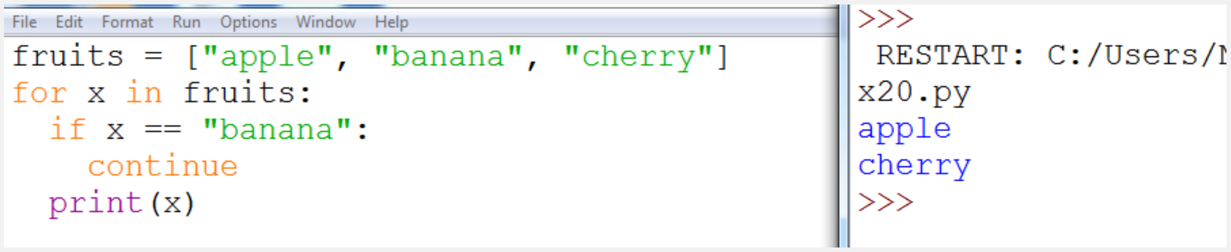
```
>>>
 RESTART: C:/Users/NIE
x20.py
apple
>>> |
```

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

for x in fruits:

  if x == "banana":

    break

  print(x)
```

The continue Statement

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

Example 20



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

```
>>>
 RESTART: C:/Users/N
x20.py
apple
cherry
>>>
```

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

Python Functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

Creating a Function

In Python a function is defined using the def keyword:

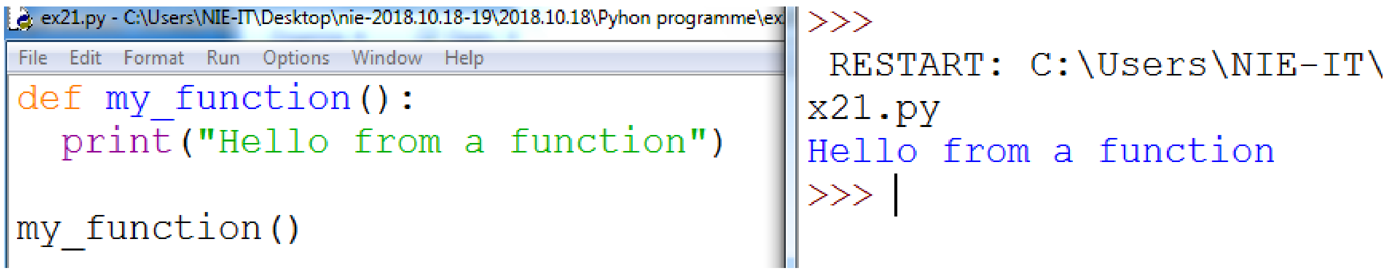
Example

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

Calling a Function

To call a function, use the function name followed by parenthesis:

Example 21


ex21.py - C:\Users\NIE-IT\Desktop\nie-2018.10.18-19\2018.10.18\Pyhon programme\ex
File Edit Format Run Options Window Help

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

my_function()
```

```
>>>
 RESTART: C:\Users\NIE-IT\
x21.py
Hello from a function
>>> 
```

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

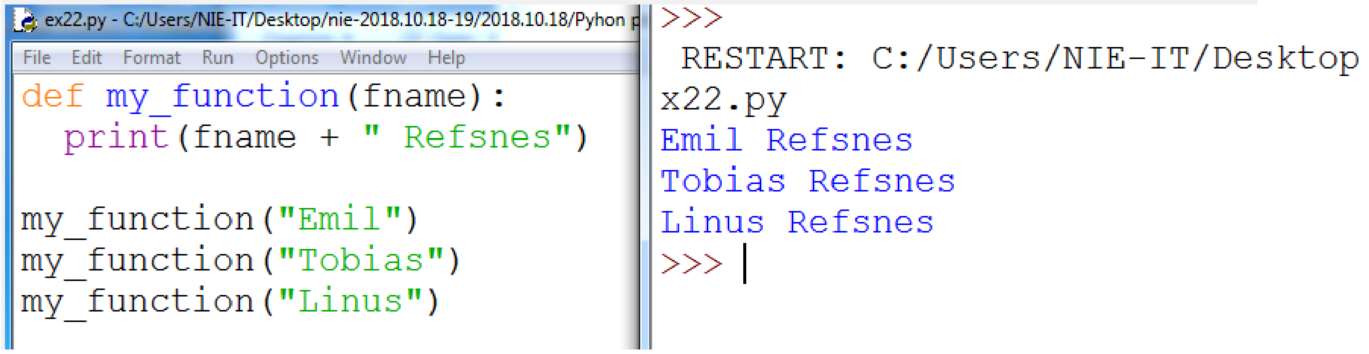
Parameters

Information can be passed to functions as parameter.

Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

The following example has a function with one parameter (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

Example 22



```
ex22.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon p
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

```
>>>
 RESTART: C:/Users/NIE-IT/Desktop
x22.py
Emil Refsnes
Tobias Refsnes
Linus Refsnes
>>>
```

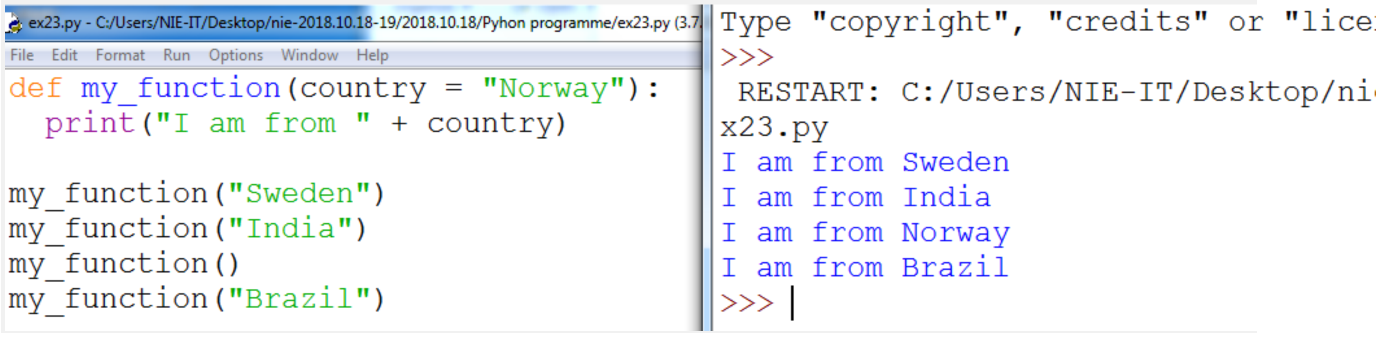
```
def my_function(fname):
  print(fname + " Refsnes")


my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

Default Parameter Value

The following example shows how to use a default parameter value.

If we call the function without parameter, it uses the default value:

Example 23



```
ex23.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex23.py (3.7.
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

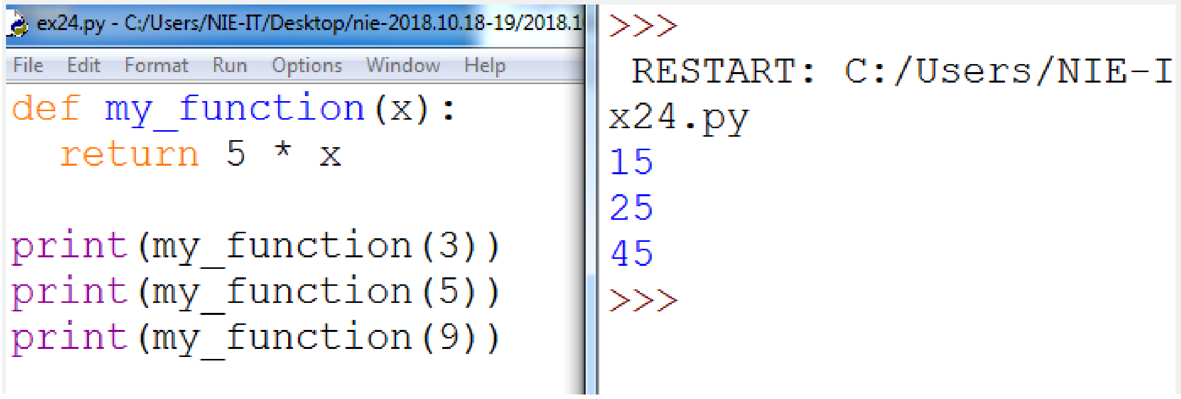
```
Type "copyright", "credits" or "lice
>>>
 RESTART: C:/Users/NIE-IT/Desktop/ni
x23.py
I am from Sweden
I am from India
I am from Norway
I am from Brazil
>>>
```

```
def my_function(country = "Norway"):
  print("I am from " + country)


my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

Return Values

To let a function return a value, use the **return** statement:

Example 24



ex24.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.1

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

```
>>>
 RESTART: C:/Users/NIE-I
x24.py
15
25
45
>>>
```

def my_function(x):

 return 5 * x

print(my_function(3))

print(my_function(5))

print(my_function(9))

Scope of Variables

All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.

The scope of a variable determines the portion of the program where you can access a particular identifier. There are two basic scopes of variables in Python −

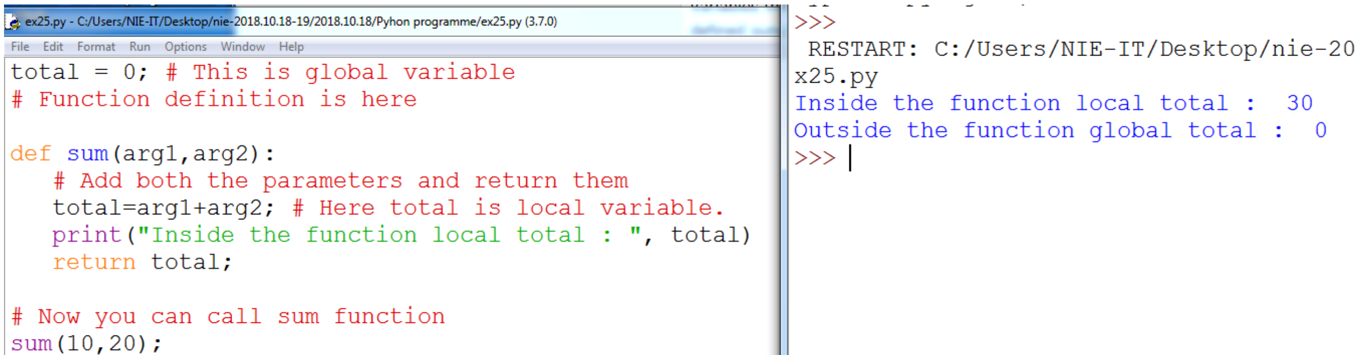
- Global variables
- Local variables

Global vs. Local variables

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope.

Example 25



ex25.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex25.py (3.7.0)

```
total = 0; # This is global variable
# Function definition is here

def sum(arg1,arg2):
   # Add both the parameters and return them
   total=arg1+arg2; # Here total is local variable.
   print("Inside the function local total : ", total)
   return total;

# Now you can call sum function
sum(10,20);
```

```
>>>
 RESTART: C:/Users/NIE-IT/Desktop/nie-20
x25.py
Inside the function local total :  30
Outside the function global total :  0
>>> |
total = 0; # This is global variable
# Function definition is here

def sum(arg1,arg2):
   # Add both the parameters and return them
   total=arg1+arg2; # Here total is local variable.
   print("Inside the function local total : ", total)
   return total;

# Now you can call sum function
sum(10,20);
print("Outside the function global total : ", total)
```

Competency 9.10 and 9.11: Python programming

- Data structures
  - Strings
  - Lists
  - Tuples
  - Dictionaries

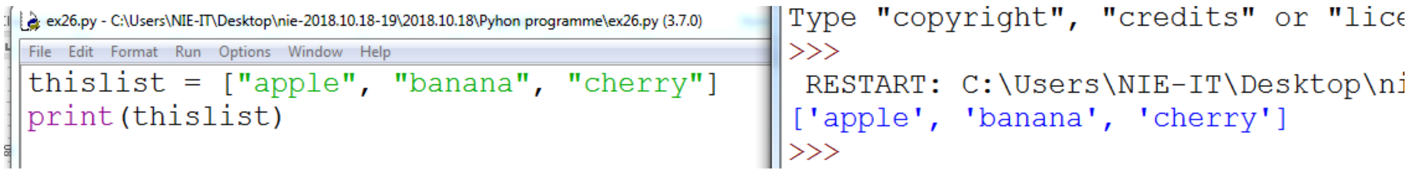
Strings – Refer previous example 10 (In Competency 9.6 and 9.7 part)

List

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

Example 26

Create a List:


ex26.py - C:\Users\NIE-IT\Desktop\nie-2018.10.18-19\2018.10.18\Pyhon programme\ex26.py (3.7.0)
File Edit Format Run Options Window Help

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

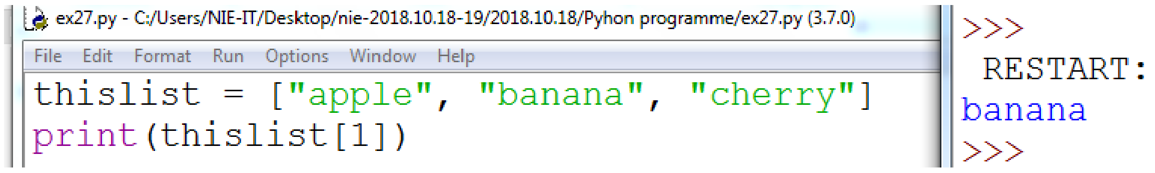
```
Type "copyright", "credits" or "lice
>>>
 RESTART: C:\Users\NIE-IT\Desktop\ni
['apple', 'banana', 'cherry']
>>>
```

thislist = ["apple", "banana", "cherry"]

print(thislist)

Example 27

Print the second item of the list:


ex27.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex27.py (3.7.0)
File Edit Format Run Options Window Help

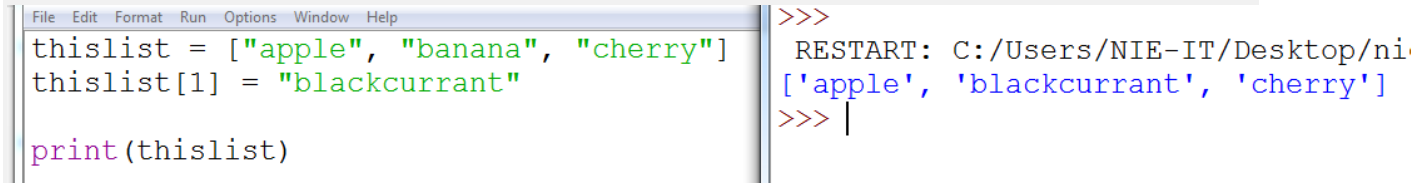
```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

```
>>>
 RESTART:
banana
>>>
```

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

Example 28

Change the second item:



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

print(thislist)
```

```
>>>
 RESTART: C:/Users/NIE-IT/Desktop/ni
['apple', 'blackcurrant', 'cherry']
>>> |
```

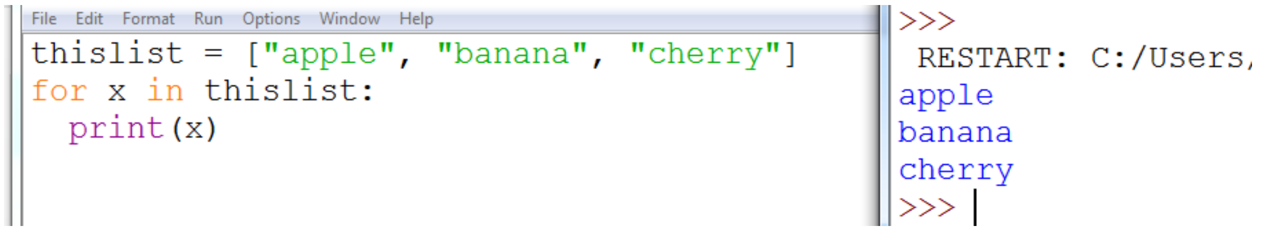
thislist = ["apple", "banana", "cherry"]

thislist[1] = "blackcurrant"

print(thislist)

Example 29

Print all items in the list, one by one:



```
File   Edit   Format   Run   Options   Window   Help
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

```
>>>
 RESTART: C:/Users,
apple
banana
cherry
>>>
```

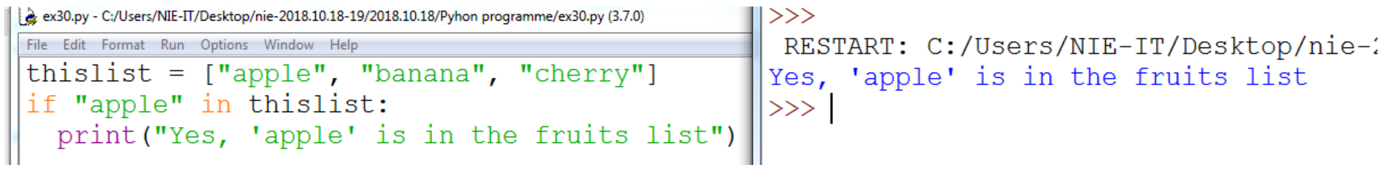
thislist = ["apple", "banana", "cherry"]

for x in thislist:

  print(x)

Example 30

Check if "apple" is present in the list:



```
ex30.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex30.py (3.7.0)
File   Edit   Format   Run   Options   Window   Help
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

```
>>>
 RESTART: C:/Users/NIE-IT/Desktop/nie-
Yes, 'apple' is in the fruits list
>>>
```

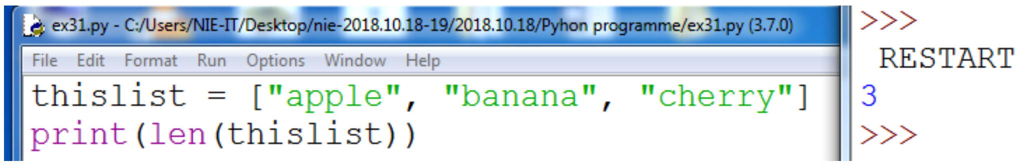
thislist = ["apple", "banana", "cherry"]

if "apple" in thislist:

  print("Yes, 'apple' is in the fruits list")

Example 31

Print the number of items in the list:



```
ex31.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex31.py (3.7.0)
File   Edit   Format   Run   Options   Window   Help
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

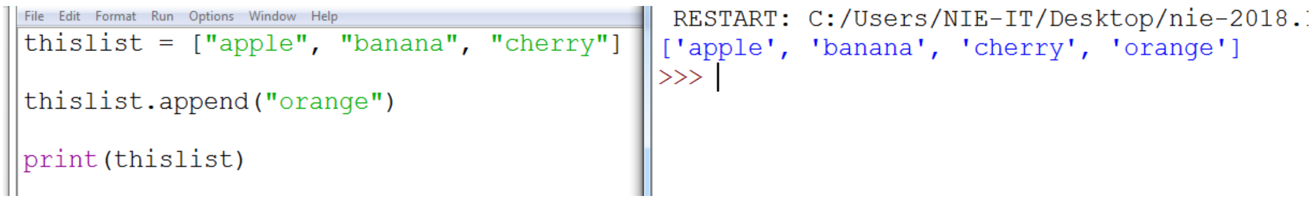
```
>>>
 RESTART
3
>>>
```

thislist = ["apple", "banana", "cherry"]

print(len(thislist))

Example 32

Using the append() method to append an item(Add Item):



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

thislist.append("orange")

print(thislist)
```

```
 RESTART: C:/Users/NIE-IT/Desktop/nie-2018.
['apple', 'banana', 'cherry', 'orange']
>>> 
```

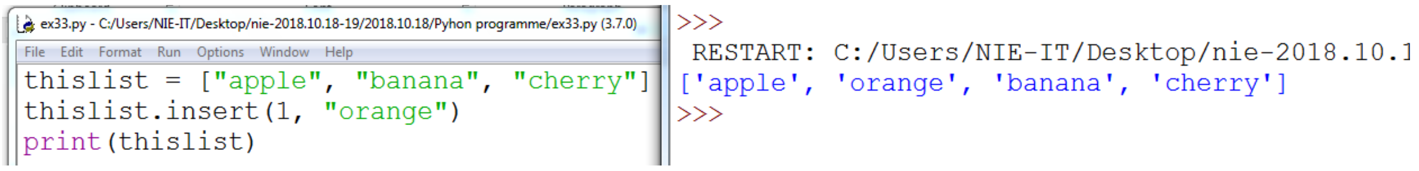
thislist = ["apple", "banana", "cherry"]

thislist.append("orange")

print(thislist)

Example 33

Insert an item as the second position:



ex33.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex33.py (3.7.0)
File Edit Format Run Options Window Help

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

```
>>>
 RESTART: C:/Users/NIE-IT/Desktop/nie-2018.10.1
['apple', 'orange', 'banana', 'cherry']
>>>
```

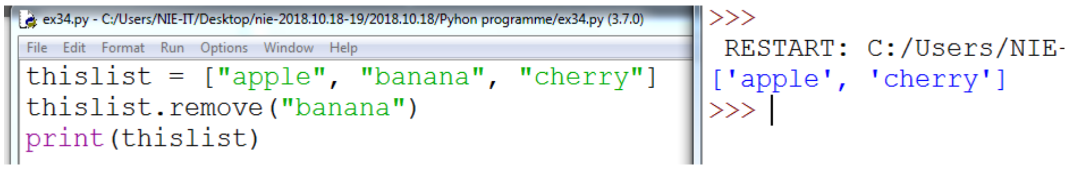
thislist = ["apple", "banana", "cherry"]

thislist.insert(1, "orange")

print(thislist)

Example 34

The remove() method removes the specified item:



ex34.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex34.py (3.7.0)
File Edit Format Run Options Window Help

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

```
>>>
 RESTART: C:/Users/NIE-
['apple', 'cherry']
>>>
```

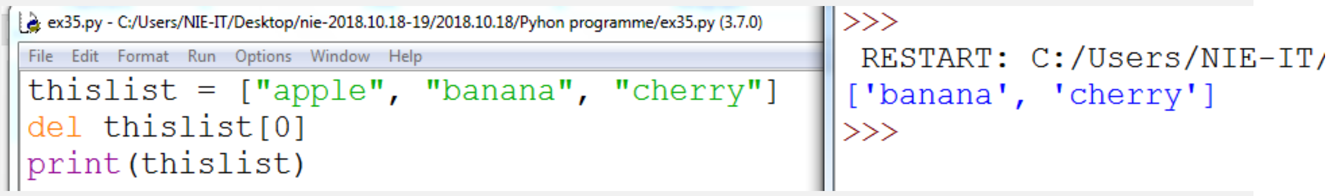
thislist = ["apple", "banana", "cherry"]

thislist.remove("banana")

print(thislist)

Example 35

The del keyword removes the specified index:



ex35.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex35.py (3.7.0)
File Edit Format Run Options Window Help

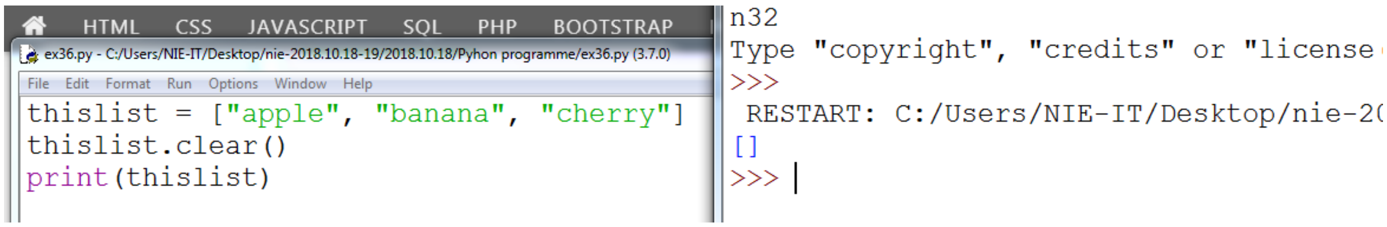
```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

```
>>>
 RESTART: C:/Users/NIE-IT/
['banana', 'cherry']
>>>
```

thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)

Example 36

The clear() method empties the list:


HTML
CSS
JAVASCRIPT
SQL
PHP
BOOTSTRAP
ex36.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex36.py (3.7.0)
File
Edit
Format
Run
Options
Window
Help

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

```
n32
Type "copyright", "credits" or "license
>>>
 RESTART: C:/Users/NIE-IT/Desktop/nie-2
[]
>>> 
```

thislist = ["apple", "banana", "cherry"]

thislist.clear()

print(thislist)

List Methods

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

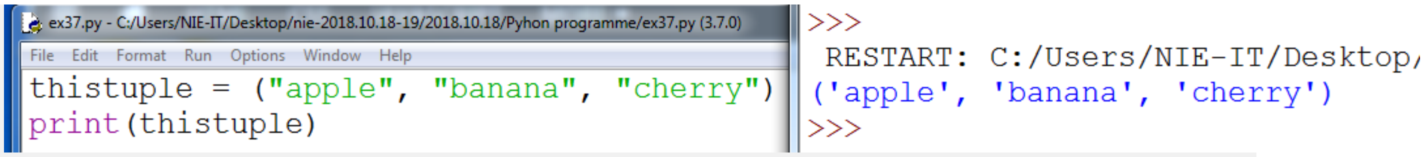
Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	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

Tuples

A tuple is a collection which is ordered and **unchangeable**. In Python tuples are written with round brackets.

Example 37

Create a Tuple:



ex37.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex37.py (3.7.0)
File Edit Format Run Options Window Help

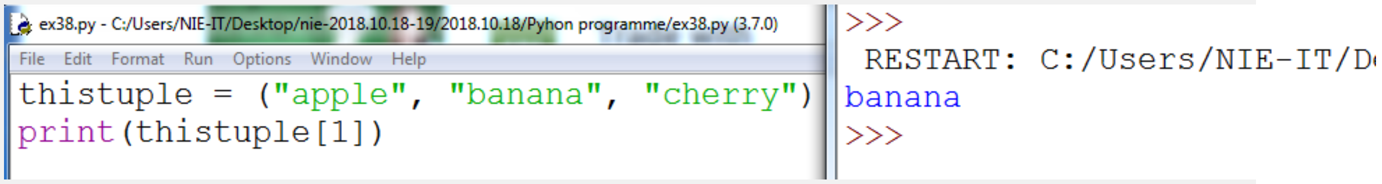
```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

```
>>>
 RESTART: C:/Users/NIE-IT/Desktop/
('apple', 'banana', 'cherry')
>>>
```

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

Example 38

Return the item in position 1:



ex38.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex38.py (3.7.0)
File Edit Format Run Options Window Help

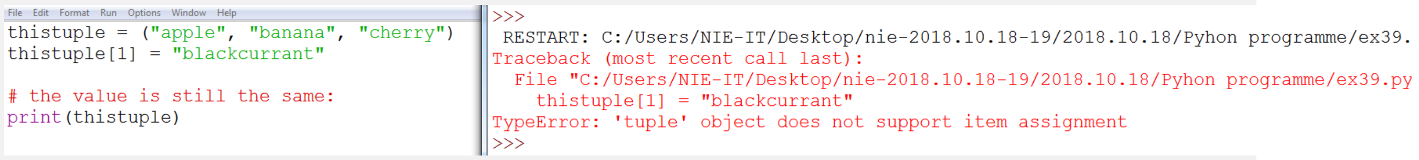
```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

```
>>>
 RESTART: C:/Users/NIE-IT/D
banana
>>>
```

thistuple = ("apple", "banana", "cherry")
print(thistuple[1])

Example 39

You cannot change values in a tuple:



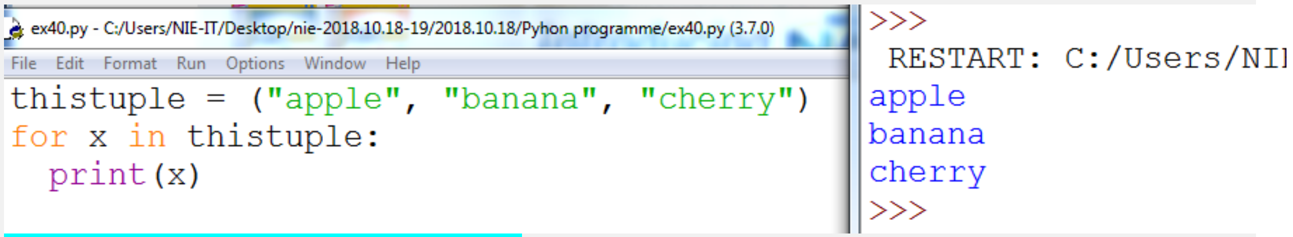
```
thistuple = ("apple", "banana", "cherry")
thistuple[1] = "blackcurrant"

# the value is still the same:
print(thistuple)
```

```
>>>
 RESTART: C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex39.
Traceback (most recent call last):
  File "C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex39.py
    thistuple[1] = "blackcurrant"
TypeError: 'tuple' object does not support item assignment
>>>
```

thistuple = ("apple", "banana", "cherry")
thistuple[1] = "blackcurrant"
# the value is still the same:
print(thistuple)

Example 40

Iterate through the items and print the values:


ex40.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex40.py (3.7.0)
File Edit Format Run Options Window Help

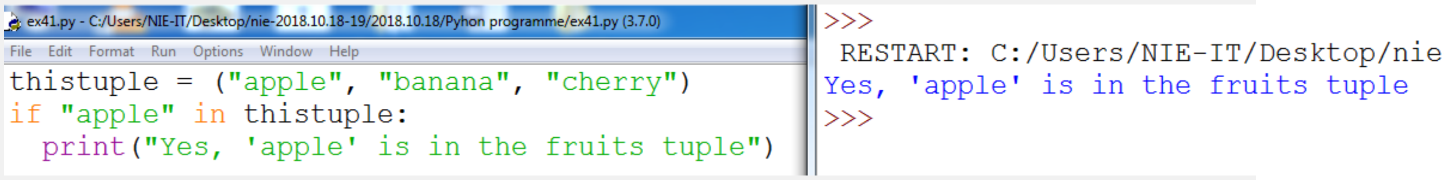
```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

```
>>>
 RESTART: C:/Users/NI
apple
banana
cherry
>>>
```

thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)

Example 41

Check if "apple" is present in the tuple:


ex41.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex41.py (3.7.0)
File Edit Format Run Options Window Help

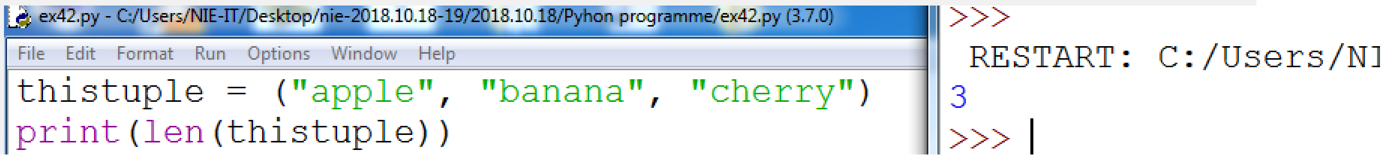
```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

```
>>>
 RESTART: C:/Users/NIE-IT/Desktop/nie
Yes, 'apple' is in the fruits tuple
>>>
```

thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")

Example 42

Print the number of items in the tuple:


ex42.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex42.py (3.7.0)
File Edit Format Run Options Window Help

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

```
>>>
 RESTART: C:/Users/NI
3
>>> |
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

- ❖ Tuples are **unchangeable**, so you cannot remove items from it, but you can delete the tuple completely.

Tuple Methods

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

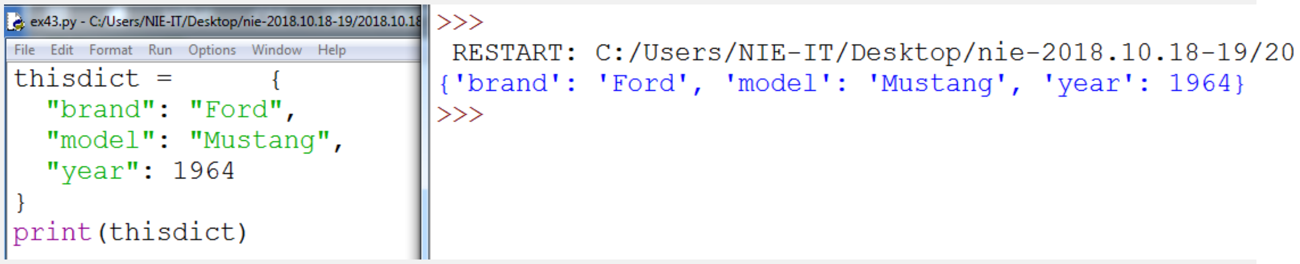
Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

Dictionaries

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

Example 43

Create and print a dictionary:


ex43.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18
File Edit Format Run Options Window Help

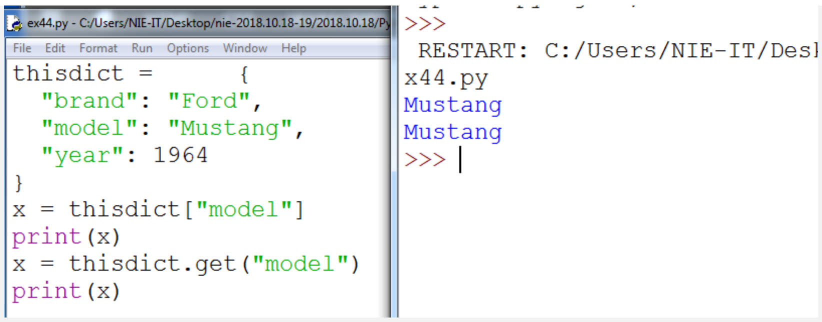
```
thisdict =       {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

```
>>>
 RESTART: C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/20
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
>>>
```

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

Example 44

Get the value of the "model" key (There is also a method called get() that will give you the same result:)


ex44.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Py
File Edit Format Run Options Window Help

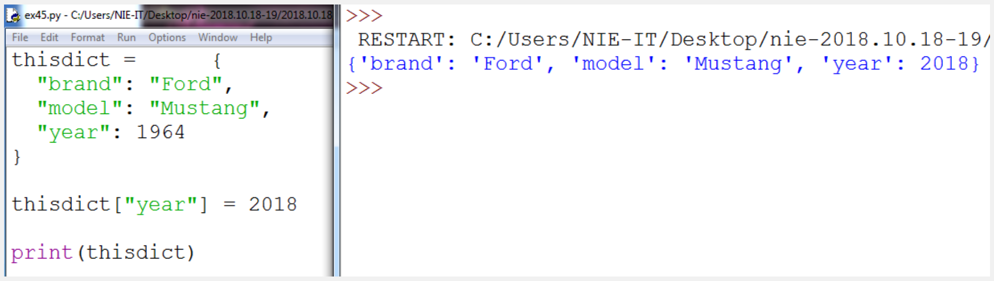
```
thisdict =       {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
print(x)
x = thisdict.get("model")
print(x)
```

```
>>>
 RESTART: C:/Users/NIE-IT/Des
x44.py
Mustang
Mustang
>>>
```

```
thisdict =       {
 "brand": "Ford",
 "model": "Mustang",
 "year": 1964
}
x = thisdict["model"]
print(x)
x = thisdict.get("model")
print(x)
```

Example 45

Change the "year" to 2018:


ex45.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18
File Edit Format Run Options Window Help

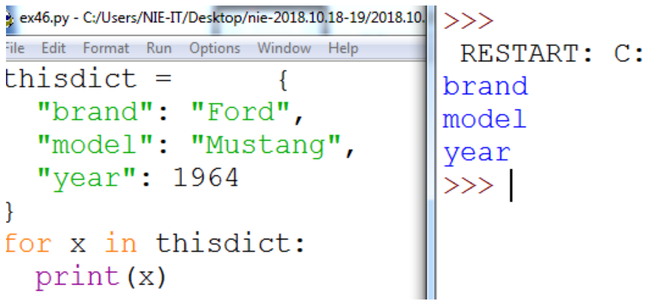
```
thisdict =       {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

thisdict["year"] = 2018

print(thisdict)
```

```
>>>
 RESTART: C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/
{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
>>>
```

```
thisdict =       {
 "brand": "Ford",
 "model": "Mustang",
 "year": 1964
}
thisdict["year"] = 2018
print(thisdict)
```

Example 46

Print all key names in the dictionary, one by one:



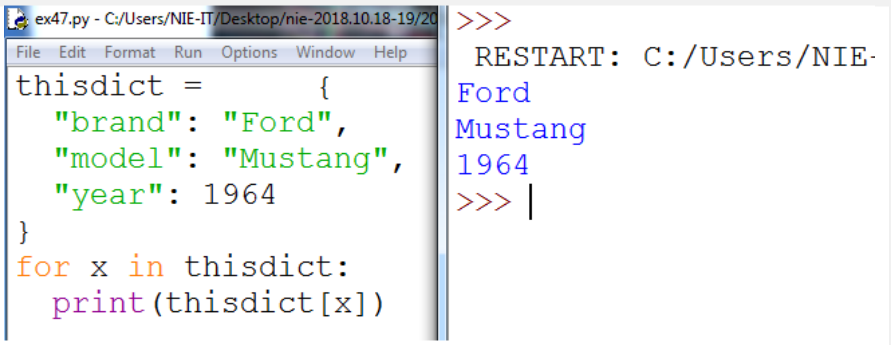
```
thisdict =          {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(x)
```

```
>>>
 RESTART: C:
brand
model
year
>>> |
```

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

Example 47

Print all *values* in the dictionary, one by one:



```
thisdict =          {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(thisdict[x])
```

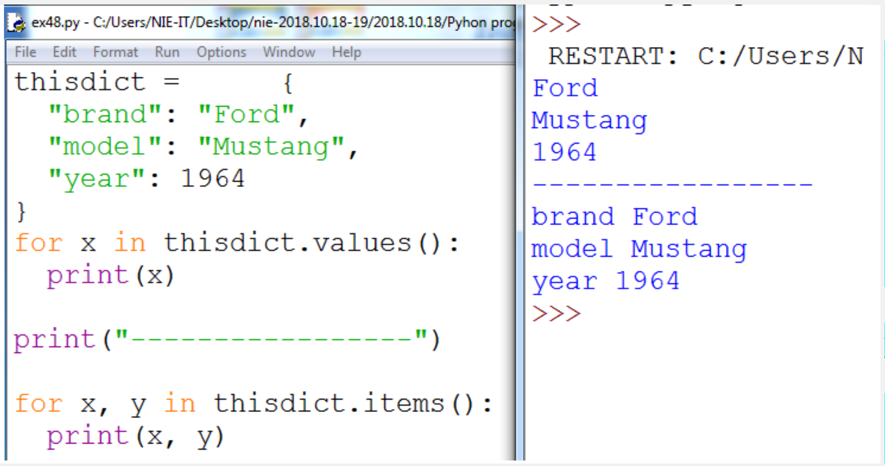
```
>>>
 RESTART: C:/Users/NIE-
Ford
Mustang
1964
>>> |
```

```
thisdict =          {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(thisdict[x])
```

Example 48

You can also use the values() function to return values of a dictionary.

Loop through both *keys* and *values*, by using the items() function:



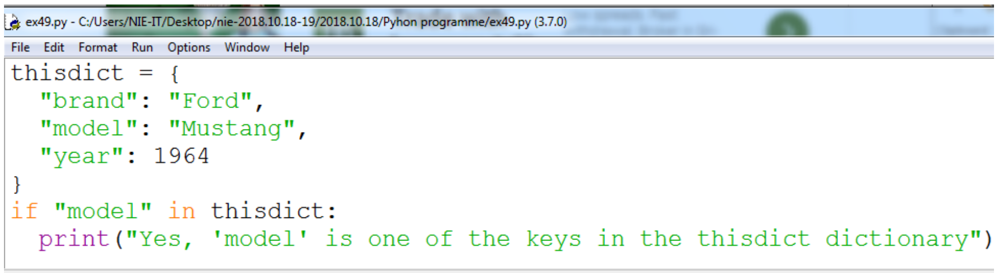
```
thisdict =          {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict.values():
  print(x)

print("-----------------")

for x, y in thisdict.items():
  print(x, y)
```

```
>>>
 RESTART: C:/Users/N
Ford
Mustang
1964
-----------------
brand Ford
model Mustang
year 1964
>>>
```

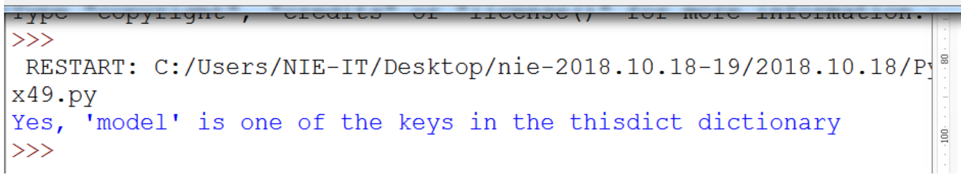
```
thisdict =          {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict.values():
  print(x)

print("-----------------")

for x, y in thisdict.items():
  print(x, y)
```

Example 49

Check if "model" is present in the dictionary (Check if Key Exists).



```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

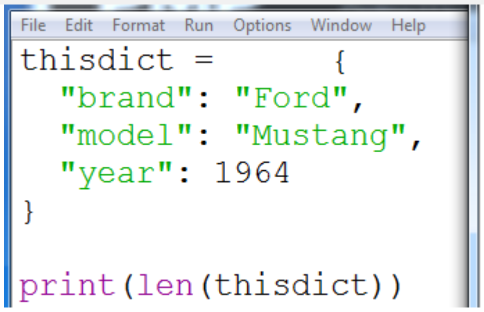


```
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Py
x49.py
Yes, 'model' is one of the keys in the thisdict dictionary
>>>
```

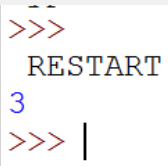
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

Example 50

Print the number of items in the dictionary:



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

print(len(thisdict))
```

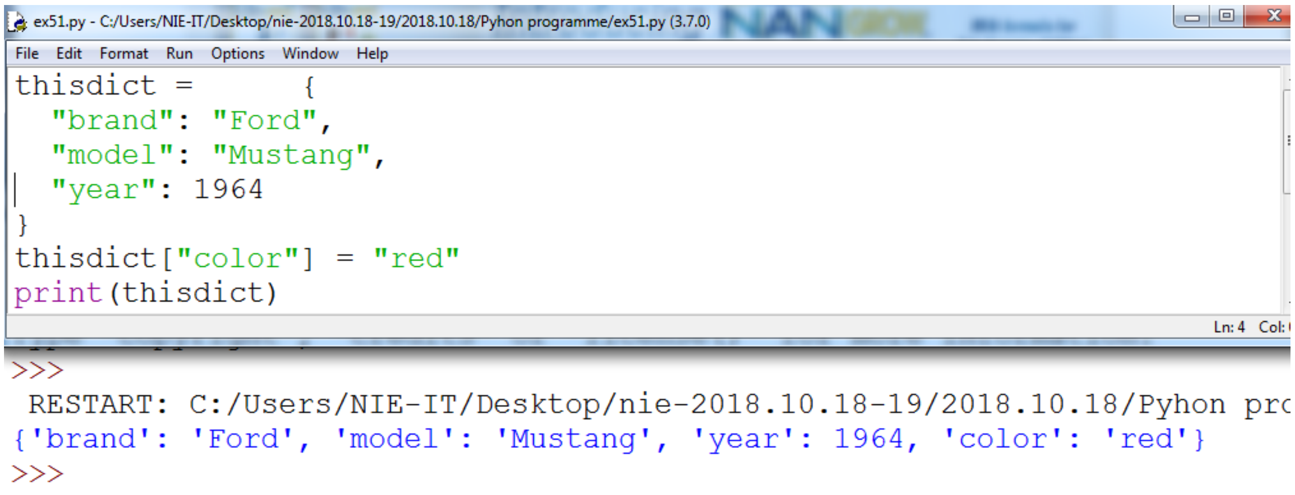


```
>>>
 RESTART
3
>>> |
```

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

print(len(thisdict))
```

Example 51

Adding an item to the dictionary is done by using a new index key and assigning a value to it.



ex51.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon programme/ex51.py (3.7.0)

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

```
>>>
 RESTART: C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyhon prc
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
>>>
```

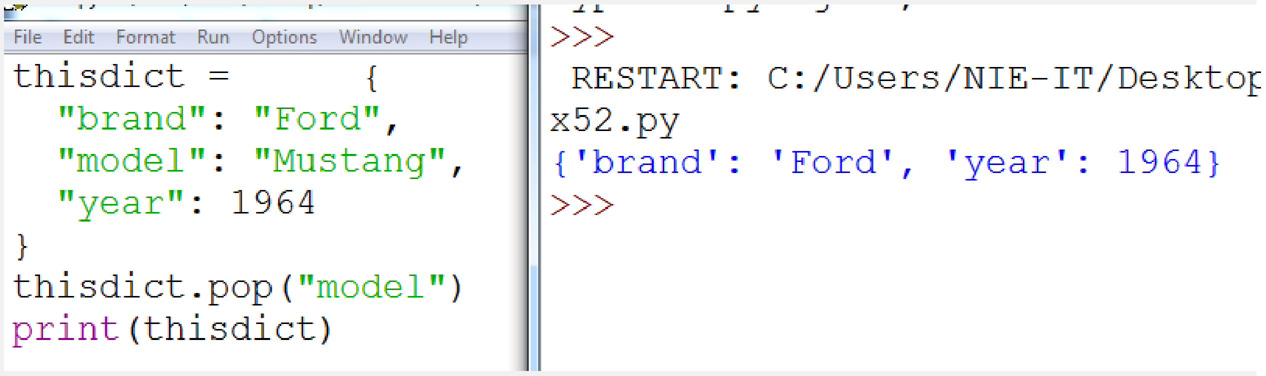
```
thisdict =      {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

Removing Items

There are several methods to remove items from a dictionary:

Example 52

The pop() method removes the item with the specified key name:



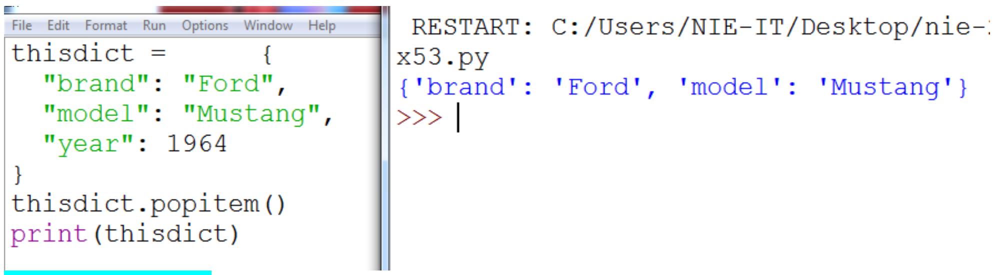
```
thisdict =      {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

```
>>>
 RESTART: C:/Users/NIE-IT/Desktop
x52.py
{'brand': 'Ford', 'year': 1964}
>>>
```

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

Example 53

The popitem() method removes the last inserted item.



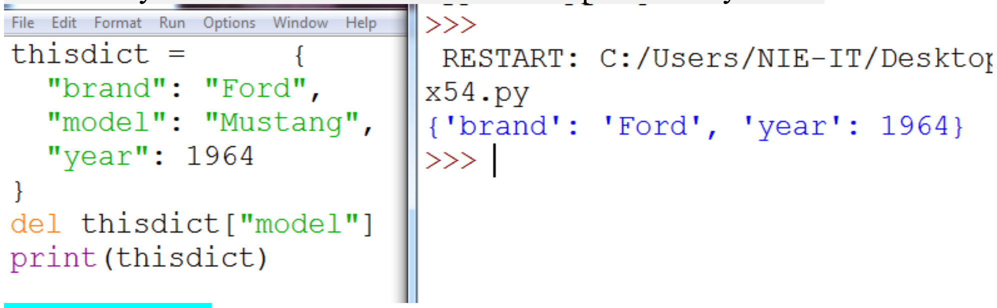
```
thisdict =       {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()
print(thisdict)
```

```
 RESTART: C:/Users/NIE-IT/Desktop/nie-
x53.py
{'brand': 'Ford', 'model': 'Mustang'}
>>> |
```

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

Example 54

The del keyword removes the item with the specified key name.



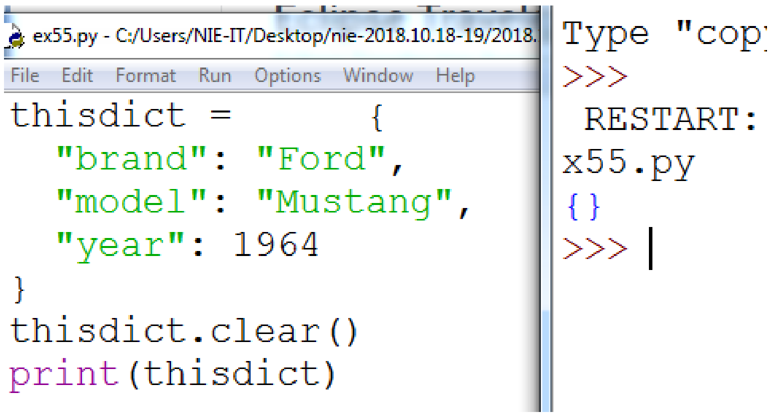
```
thisdict =       {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

```
>>>
 RESTART: C:/Users/NIE-IT/Desktop
x54.py
{'brand': 'Ford', 'year': 1964}
>>> |
```

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

Example 55

The `clear()` keyword empties the dictionary


ex55.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.
File Edit Format Run Options Window Help

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

```
Type "cop
>>>
 RESTART:
x55.py
{}
>>> 
```

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

Dictionary Methods

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

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and values
get()	Returns the value of the specified key
items()	Returns a list containing the a tuple for each key value pair
keys()	Returns a list contianing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Python File handling

- Python has several functions for creating, reading, updating, and deleting files.
- Uses basic file operations (open, close, read write and append)
- The key function for working with files in Python is the open() function.

The open() function takes two parameters; *filename*, and *mode*.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

```
"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists
```

In addition you can specify if the file should be handled as binary or text mode

```
"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)
```

Syntax

To open a file for reading it is enough to specify the name of the file:

f = open("n1.txt")

The code above is the same as:

f = open("n1.txt", "rt")

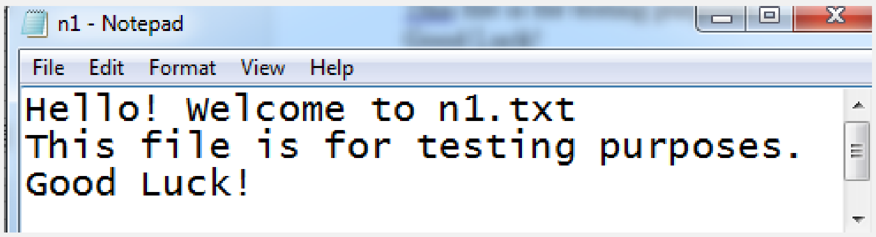
Because "r" for read, and "t" for text are the default values, you do not need to specify them.

Note: Make sure the file exists, or else you will get an error.

Open a File on the Server

Assume we have the following file, located in the same folder as Python:

n1.txt

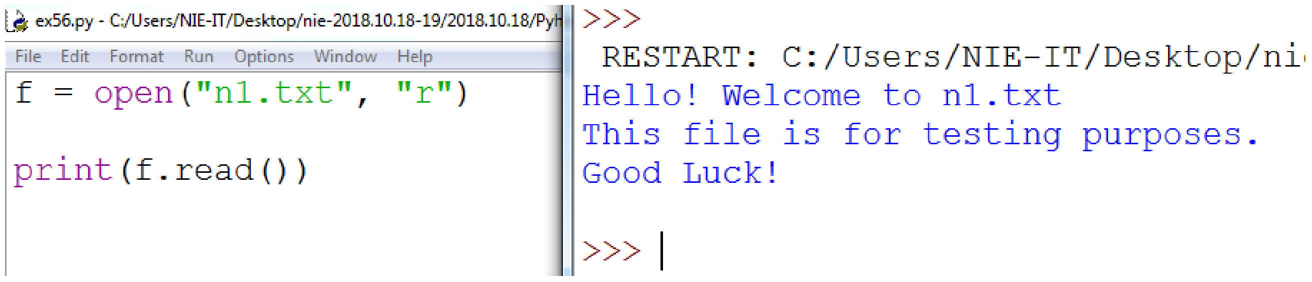


```
Hello! Welcome to n1.txt
This file is for testing purposes.
Good Luck!
```

To open the file, use the built-in open() function.

The open() function returns a file object, which has a read() method for reading the content of the file:

Example 56


ex56.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18/Pyt
File Edit Format Run Options Window Help

```
f = open("n1.txt", "r")

print(f.read())
```

```
>>>
 RESTART: C:/Users/NIE-IT/Desktop/ni
Hello! Welcome to n1.txt
This file is for testing purposes.
Good Luck!

>>> |
```

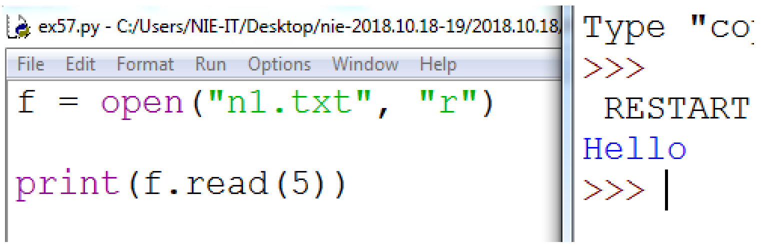
f = open("n1.txt", "r")
print(f.read())

Read Only Parts of the File

By default the `read()` method returns the whole text, but you can also specify how many character you want to return:

Example 57

Return the 5 first characters of the file:


ex57.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10.18
File Edit Format Run Options Window Help

```
f = open("n1.txt", "r")

print(f.read(5))
```

```
Type "co
>>>
 RESTART
Hello
>>> 
```

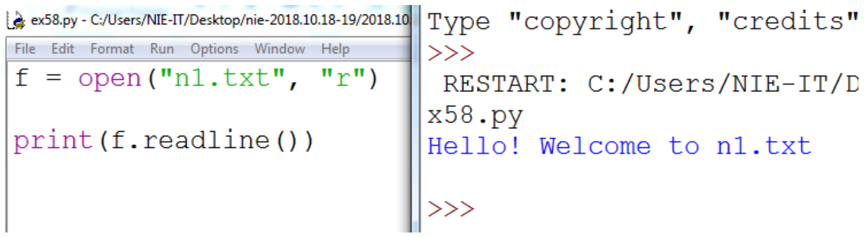
f = open("n1.txt", "r")
print(f.read(5))

Read Lines

You can return one line by using the readline() method:

Example 58

Read one line of the file:

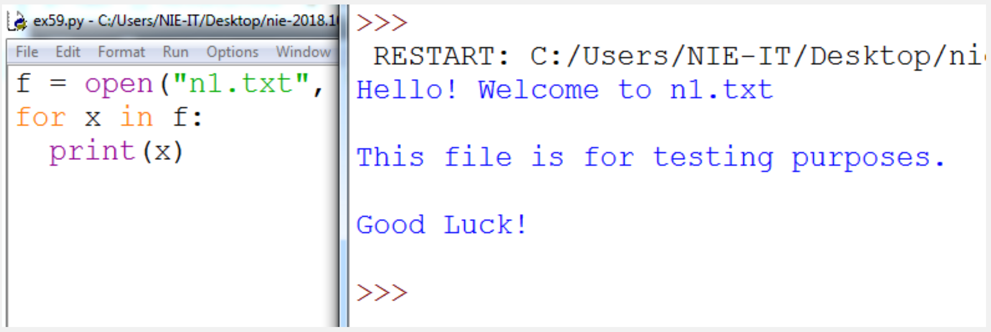

ex58.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.10
File Edit Format Run Options Window Help

```
f = open("n1.txt", "r")

print(f.readline())
```

```
Type "copyright", "credits"
>>>
 RESTART: C:/Users/NIE-IT/D
x58.py
Hello! Welcome to n1.txt

>>>
```

f = open("n1.txt", "r")
print(f.readline())

Example 59


ex59.py - C:/Users/NIE-IT/Desktop/nie-2018.1
File Edit Format Run Options Window

```
f = open("n1.txt",
for x in f:
  print(x)
```

```
>>>
 RESTART: C:/Users/NIE-IT/Desktop/ni
Hello! Welcome to n1.txt

This file is for testing purposes.

Good Luck!

>>>
```

f = open("n1.txt", "r")
for x in f:
  print(x)

Write to an Existing File

To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

Open the file "n1.txt" and append content to the file:

Example

Open the file "n1.txt" and append content to the file:

```
f = open("n1.txt", "a")
f.write("Now the file has one more line!")
```

Example

Open the file "n1.txt" and overwrite the content:

```
f = open("n1.txt", "w")
f.write("Woops! I have deleted the content!")
```

Note: the "w" method will overwrite the entire file.

Create a New File

To create a new file in Python, use the open() method, with one of the following parameters:

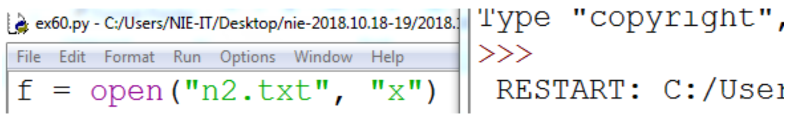
"x" - Create - will create a file, returns an error if the file exist

"a" - Append - will create a file if the specified file does not exist

"w" - Write - will create a file if the specified file does not exist

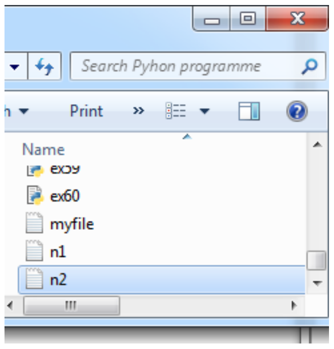
Example 60

Create a file called "n2.txt":


ex60.py - C:/Users/NIE-IT/Desktop/nie-2018.10.18-19/2018.
File Edit Format Run Options Window Help

```
f = open("n2.txt", "x")
```

```
Type "copyright",
>>>
 RESTART: C:/Use
```



```
f = open("n2.txt", "x")
```

Delete a File

To delete a file, you must import the OS module, and run its `os.remove()` function:

Example

Remove the file "n1.txt":

```
import os
os.remove("n1.txt")
```

Check if File exist:

To avoid getting an error, you might want to check if the file exist before you try to delete it:

Example

Check if file exist, *then* delete it:

```
import os
if os.path.exists("n1.txt"):
  os.remove("n1.txt")
else:
  print("The file does not exist")
```

Delete Folder

To delete an entire folder, use the `os.rmdir()` method:

Example

Remove the folder "myfolder":

```
import os
os.rmdir("myfolder")
```

Note: You can only remove *empty* folders

Competency 9.12 and 9.13: Python programming

Introduction to SQL

SQL is a standard language for accessing and manipulating databases.

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

Using SQL in Your Web Site

To build a web site that shows data from a database, you will need:

- An RDBMS database program (i.e. MS Access, SQL Server, MySQL)
- To use a server-side scripting language, like PHP or ASP
- To use SQL to get the data you want
- To use HTML / CSS to style the page

RDBMS

RDBMS stands for **R**elational **D**ata**b**ase **M**anagement **S**ystem.

RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.

Python MySQL

Python can be used in database applications.

One of the most popular database is MySQL.

MySQL Database

To be able experiment with the code examples, you should have MySQL installed on your computer.

MySQL database at https://www.mysql.com/downloads/.

Or

You can download a free WAMP server at https://sourceforge.net/projects/wampserver/

Or

You can download a free XAMPP server at https://www.apachefriends.org/index.html

Some of The Most Important SQL Commands

- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table
- DROP TABLE - deletes a table
- INSERT INTO - inserts new data into a database
- SELECT - retrieve data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index

Install MySQL Driver

Python needs a MySQL driver to access the MySQL database. "MySQL Connector" driver can be used for that. PIP can be used to install "MySQL Connector". PIP is a package manager for Python packages, or modules. PIP is most likely already installed in Python environment (When installing a Python, select “Customize installation” instead of “Install Now”. Then PIP will be installed automatically).

Navigate the command line to the location of Python's script directory, and type the following to check PIP is installed.

```
C:\Users\your name\AppData\Local\Programs\Python\Python37-32\Scripts> pip --version
```

If PIP is not installed, download and install it from this page: https://pypi.org/project/pip/

Navigate the command line to the location of Python's script directory, and type the following to Download and install "MySQL Connector":

```
C:\Users\your name\AppData\Local\Programs\Python\Python37-32\Scripts>pip install mysql-connector
```

Test MySQL Connector

To test if the installation was successful, or if you already have "MySQL Connector" installed, create a Python page with the following content. If the below code was executed with no errors, "MySQL Connector" is installed and ready to be used.

```
import mysql.connector
```

Create Connection

Use the username and password of MySQL database to create a connection.

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  passwd="yourpassword"
)
```

Normally default username is “root” and no password.

MySQL Create Database

Example 61

Create a database named "school"

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  passwd="yourpassword"
)

mycursor = mydb.cursor()

mycursor.execute("CREATE DATABASE school")
```

MySQL delete Database

Example 62

Delete school database

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  passwd="yourpassword"
)

mycursor = mydb.cursor()

mycursor.execute("DROP DATABASE school")
```

MySQL Create Table

Example 63
Create student table with regNo, name, address and contactNo fields.
Define the name of the database when creating the connection

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  passwd="yourpassword",

  database="school"
)

mycursor = mydb.cursor()
mycursor.execute("CREATE TABLE student (

                                        regNo VARCHAR(10) NOT NULL,

                                        name VARCHAR(150) NOT NULL,

                                        address VARCHAR(250),
```

MySQL Modify Table

Example 64
Add new field called “dob” to the student table

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  passwd="yourpassword",

  database="school"
)

mycursor = mydb.cursor()
mycursor.execute("ALTER TABLE student ADD COLUMN dob DATE")
```

MySQL Drop Table

Example 65

Delete student table

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  passwd="yourpassword",

  database="school"
)
mycursor = mydb.cursor()
mycursor.execute("DROP TABLE student")
```

MySQL Insert data into the Table

To fill a table in MySQL, "INSERT INTO" statement is used. ***mydb.commit()*** is required to make the changes, otherwise no changes are made the table.

Example 66

Insert a record in the "student" table

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  passwd="yourpassword",

  database="school"
)
mycursor = mydb.cursor()
sql = "INSERT INTO student (regNo, name, address, contactNo, dob) VALUES (%s, %s, %s, %s, %s)"
val = ("1200", "John", "Highway 21", "0715623410", "2000-10-21")
mycursor.execute(sql, val)
```

```
1 record inserted.
```

Example 67

Insert Multiple Rows

To insert multiple rows into a table, use the ***executemany()*** method. The second parameter of the executemany() method is a list of tuples, containing the data that want to insert.

```
sql = "INSERT INTO student (regNo, name, address, contactNo, dob) VALUES (%s, %s, %s, %s, %s)"


val = [
  ('1201', 'Peter', 'Lowstreet 4', '0715874510', '2000-06-21'),
  ('1202', 'William', 'Central st 954', '0775857410', '2000-03-11'),
  ('1203', 'Viola', 'Sideway 1633', '0710055210', '2000-08-09')
]

mycursor.executemany(sql, val)

mydb.commit()

[illegible]
```

```
3 was inserted.
```

MySQL Select Data

1) Select all records
2) Select specific columns
3) Select with filter
4) Sort the result

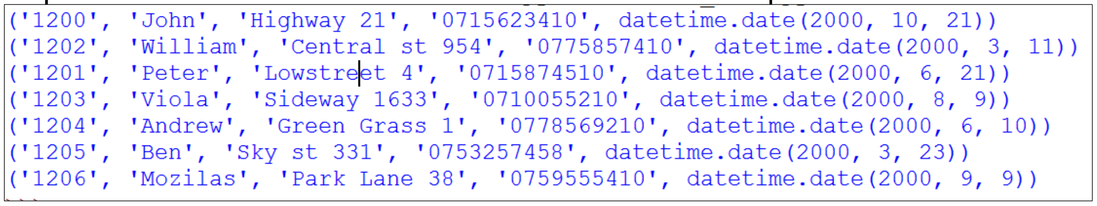
To select from a table in MySQL, use the "SELECT" statement. ***fetchall()*** method fetches all rows from the last executed statement. ***fetchone()*** method fetches only one row.

Select all records

Example 68

Select all records from the "student" table, and display the result.

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  passwd="yourpassword",
  database="school"
)

mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM student")
myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```



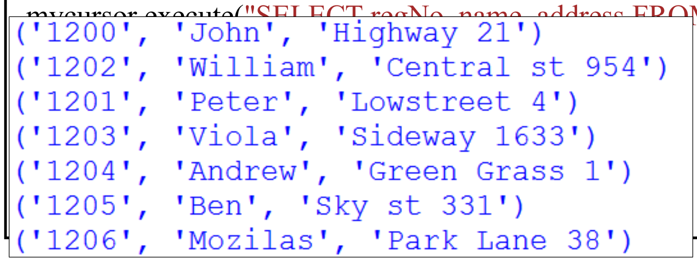
```
('1200', 'John', 'Highway 21', '0715623410', datetime.date(2000, 10, 21))
('1202', 'William', 'Central st 954', '0775857410', datetime.date(2000, 3, 11))
('1201', 'Peter', 'Lowstreet 4', '0715874510', datetime.date(2000, 6, 21))
('1203', 'Viola', 'Sideway 1633', '0710055210', datetime.date(2000, 8, 9))
('1204', 'Andrew', 'Green Grass 1', '0778569210', datetime.date(2000, 6, 10))
('1205', 'Ben', 'Sky st 331', '0753257458', datetime.date(2000, 3, 23))
('1206', 'Mozilas', 'Park Lane 38', '0759555410', datetime.date(2000, 9, 9))
```

Select specific columns

Example 69

Select only the regNo, name and address columns:

```
mycursor.execute("SELECT regNo, name, address FROM student")
```



```
('1200', 'John', 'Highway 21')
('1202', 'William', 'Central st 954')
('1201', 'Peter', 'Lowstreet 4')
('1203', 'Viola', 'Sideway 1633')
('1204', 'Andrew', 'Green Grass 1')
('1205', 'Ben', 'Sky st 331')
('1206', 'Mozilas', 'Park Lane 38')
```

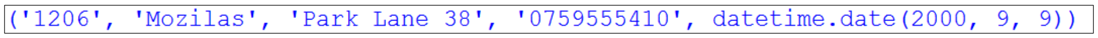
Select with filter

When selecting records from a table, selection can be filtered by using the "WHERE" statement.

Example 70

Select record(s) where the birthday is “2000-09-09 “.

```
mycursor.execute("SELECT * FROM student WHERE dob='2000-09-09 ' ")
```

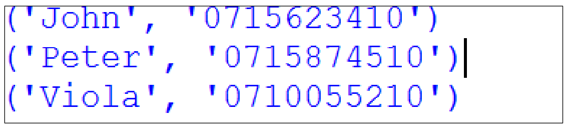


```
('1206', 'Mozilas', 'Park Lane 38', '0759555410', datetime.date(2000, 9, 9))
```

Example 71

Select records where the contactNo start with the"071"

```
mycursor.execute("SELECT name, contactNo FROM student WHERE contactNo LIKE '071%' ")
```



```
('John', '0715623410')
('Peter', '0715874510')
('Viola', '0710055210')
```

Sort the result

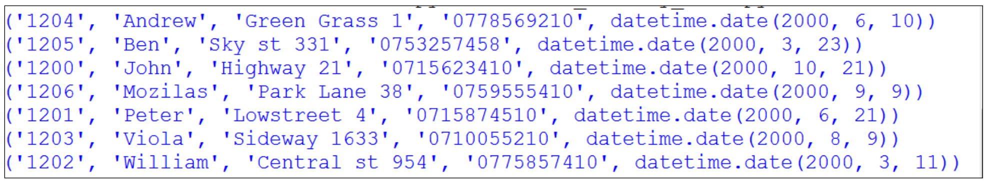
Use the ORDER BY statement to sort the result in ascending or descending order.

The ORDER BY keyword sorts the result ascending by default. To sort the result in descending order, use the DESC keyword.

Example 72

Sort the result alphabetically by name.

```
mycursor.execute("SELECT *  FROM student ORDER BY name ")
```

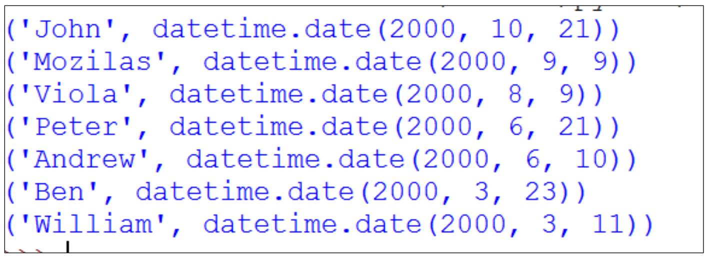


```
('1204', 'Andrew', 'Green Grass 1', '0778569210', datetime.date(2000, 6, 10))
('1205', 'Ben', 'Sky st 331', '0753257458', datetime.date(2000, 3, 23))
('1200', 'John', 'Highway 21', '0715623410', datetime.date(2000, 10, 21))
('1206', 'Mozilas', 'Park Lane 38', '0759555410', datetime.date(2000, 9, 9))
('1201', 'Peter', 'Lowstreet 4', '0715874510', datetime.date(2000, 6, 21))
('1203', 'Viola', 'Sideway 1633', '0710055210', datetime.date(2000, 8, 9))
('1202', 'William', 'Central st 954', '0775857410', datetime.date(2000, 3, 11))
```

Example 73

Display birthday in descending order.

```
mycursor.execute("SELECT name, dob  FROM student ORDER BY dob DESC ")
```



```
('John', datetime.date(2000, 10, 21))
('Mozilas', datetime.date(2000, 9, 9))
('Viola', datetime.date(2000, 8, 9))
('Peter', datetime.date(2000, 6, 21))
('Andrew', datetime.date(2000, 6, 10))
('Ben', datetime.date(2000, 3, 23))
('William', datetime.date(2000, 3, 11))
```

MySQL Update Table

Existing records can be updated in a table by using the "UPDATE" statement.

Example 74

Overwrite the name column from "Viola" to "Mathias".

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  passwd="yourpassword",
  database="school"
)

mycursor = mydb.cursor()
sql = "UPDATE student SET name='Mathias' WHERE name='Viola'  "

mycursor.execute(sql)

mydb.commit()

print(mycursor.rowcount, "record(s) affected")
```

```
1 record(s) affected
```

MySQL Delete

Records can be deleted from an existing table by using the "DELETE FROM" statement.

Example 75

Delete any record where the address is "Park Lane 38".

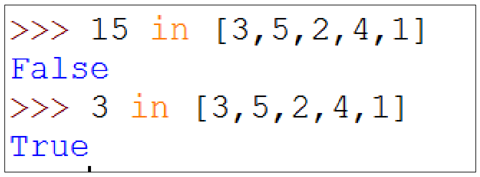
```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  passwd="yourpassword",
  database="school"
)

mycursor = mydb.cursor()
sql = "DELETE FROM student WHERE address=' Park Lane 38'  "

mycursor.execute(sql)

mydb.commit()

print(mycursor.rowcount, "record(s) deleted")
```

```
1 record(s) deleted
```

Searching Techniques

Searching is the algorithmic process of finding a particular item in a collection of items. A search typically answers either *True* or *False* as to whether the item is present. On occasion it may be modified to return where the item is found.

In Python, there is a very easy way to ask whether an item is in a list of items. We use the *in* operator.



```
>>> 15 in [3,5,2,4,1]
False
>>> 3 in [3,5,2,4,1]
True
```

Sequential Search

When data items are stored in a collection such as a list, they have a linear or sequential relationship. Each data item is stored in a position relative to the others. In Python lists, these relative positions are the index values of the individual items. Since these index values are ordered, it is possible to visit them in sequence. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data structure.

Example 75

Write a function which implements sequential search. It should take a list and an element as a parameter, and return true. If the element is not in the list, the function should return false. If the element is in the list multiple times, the function should return true when the first match is found.

```
def sequentialSearch(alist, item):
        pos = 0
        found = False

        while pos < len(alist):
                if alist[pos] == item:
                        found = True
                        break
                else:
                        pos = pos+1
```

```
False
True
```

Sorting Techniques

Sorting refers to arranging data in a particular format. It can be ascending or descending order.
Sorting algorithm specifies the way to arrange data in a particular order.

Different types of sorting algorithms

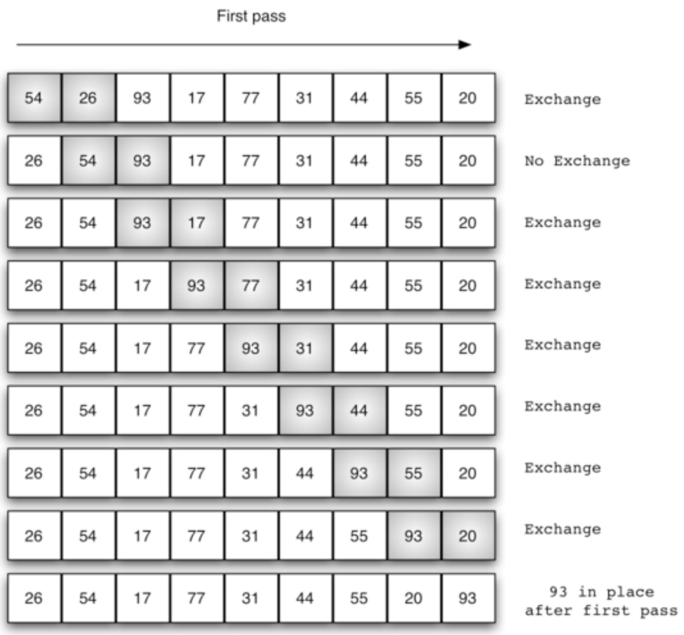
1. Bubble sort
2. Selection Sort
3. Insertion Sort
4. Shell sort

Bubble sort

The bubble sort makes multiple passes through a list. It compares adjacent items and exchanges those that are out of order. Each pass through the list places the next largest value in its proper place. In essence, each item "bubbles" up to the location where it belongs.

Sorting list in ascending order.

- compare 1st and 2nd elements
- if 1st larger than 2nd, swap
- compare 2nd and 3rd, and swap if necessary
- continue until compare the last two elements
- largest element is now the last element in the array.
- repeat starting from the beginning until no swaps are performed (i.e., the list is sorted)
- each time you go through the elements bubbling up the largest element
- no need to try the last i elements for the ith run since the end elements are already sorted



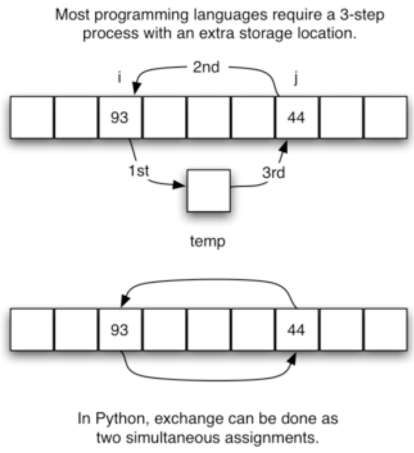
First pass
54 26 93 17 77 31 44 55 20 Exchange
26 54 93 17 77 31 44 55 20 No Exchange
26 54 93 17 77 31 44 55 20 Exchange
26 54 17 93 77 31 44 55 20 Exchange
26 54 17 77 93 31 44 55 20 Exchange
26 54 17 77 31 93 44 55 20 Exchange
26 54 17 77 31 44 93 55 20 Exchange
26 54 17 77 31 44 55 93 20 Exchange
26 54 17 77 31 44 55 20 93 93 in place after first pass

At the start of the second pass, the largest value is now in place. There are n−1n−1 items left to sort, meaning that there will be n−2n−2 pairs. Since each pass places the next largest value in place, the total number of passes necessary will be n−1n−1. After completing the n−1n−1 passes, the smallest item must be in the correct position with no further processing required. bubbleSort function shows the complete bubble sort function. It takes the list as a parameter, and modifies it by exchanging items as necessary.

The exchange operation, sometimes called a "swap," is slightly different in Python than in most other programming languages. Typically, swapping two elements in a list requires a temporary storage location (an additional memory location). A code fragment such as

```
temp = alist[i]
alist[i] = alist[j]
alist[j] = temp
```

It will exchange the i^{th} and j^{th} items in the list. Without the temporary storage, one of the values would be overwritten. In Python, it is possible to perform simultaneous assignment. The statement a,b=b,a will result in two assignment statements being done at the same time (see Figure 2). Using simultaneous assignment, the exchange operation can be done in one statement.


Most programming languages require a 3-step process with an extra storage location.
i
2nd
j
93
44
1st
3rd
temp
93
44
In Python, exchange can be done as two simultaneous assignments.

```
def bubbleSort(alist):

    for num in range(len(alist)-1,0,-1):

        for i in range(num):
            if alist[i]>alist[i+1]:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp

alist = [54,26,93,17,77,31,44,55,20]

bubbleSort(alist)
```

```
def bubbleSort(alist):

    for num in range(len(alist)-1,0,-1):

        for i in range(num):
            if alist[i]>alist[i+1]:
                alist[i], alist[i+1] = alist[i+1] , alist[i]

alist = [54,26,93,17,77,31,44,55,20]

bubbleSort(alist)
```

```
[17, 20, 26, 31, 44, 54, 55, 77, 93]
```