Python programming applied to Finance – 1

Jean-Damien VILLIERS

1-Introduction

Python programming applied to Finance

Python 1:

- Fundamentals for python
- Introduction to Data Science tools for Finance
- Python for reporting, automation of tasks, and interaction with Excel/VBA

Python 2:

- Advanced tools for Data Science
- Object oriented programming
- Setting up your own python server / cloud
- Project applied to Quantitative Finance

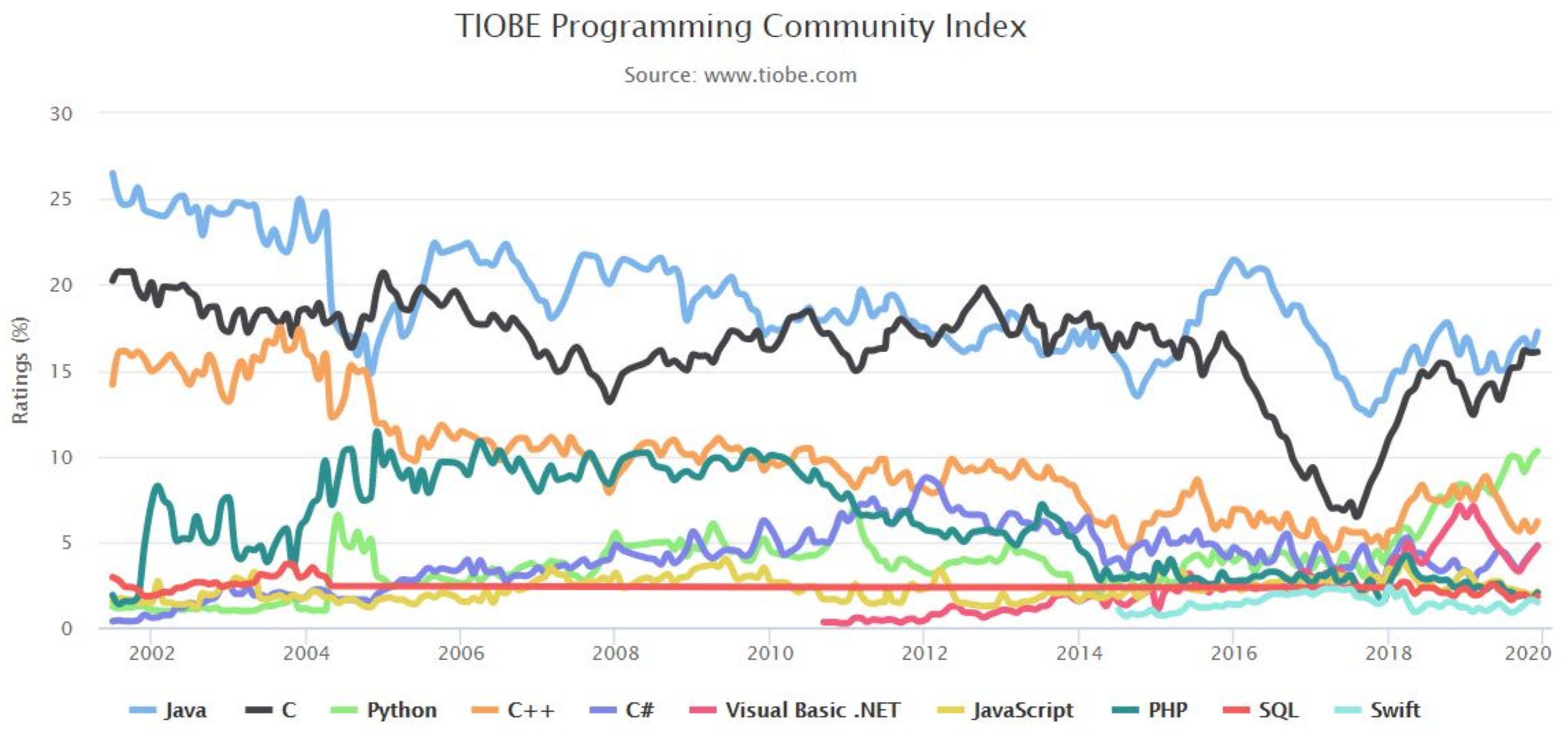
Exam

- 1h30 on computer

- Multiple choice questions

  - Detecting python syntax issues
  - Writing pieces of code from scratch
  - Main definitions

Evolution of programming language ranking



Programming languages used in finance

- **Pricing Libraries:**
  C++, ADA: object oriented languages
  Fast, mix of high and low level feature

- **Desktop software / backend programs:**
  Java / Jython
  Platform independence, object oriented, adequate for GUI

- **Calibration, backtesting, data analysis:**
  Python / R
  Simple, “scripting” language, for both beginner and advanced developers

- **Reporting, day to day tools:**
  Excel / VBA / Python

2- Development tools

Development tools

- **Editors**

An editor allows the programmer to enter the program source code and save it to files. Most programming editors increase programmer productivity by using colors to highlight language features.

- **Compilers**

A compiler translates the source code to target code. The target code may be the machine language for a particular platform or embedded device. The target code could be another source language.

- **Interprers**

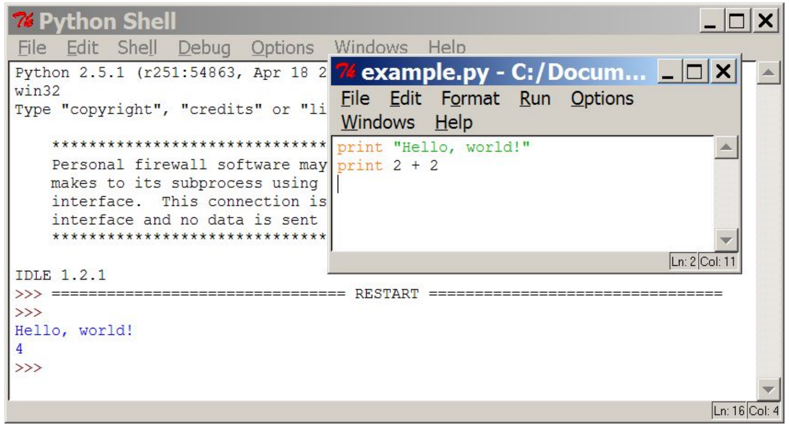
Like a compiler, in that it translates higher-level source code into target code (usually machine language). However, while a compiler produces an executable program that may run many times with no additional translation needed, an interpreter translates source code statements into machine language each time a user runs the program. A compiled program does not need to be recompiled to run, but an interpreted program must be reinterpreted each time it executes. For this reason interpreted languages are often referred to as scripting languages.

- **IDE**

Integrated Development Environment. An IDE includes editors, debuggers, and other programming aids in one comprehensive program.
Example of Python IDE: Spyder, Jetbrains, Jupyter Notebook

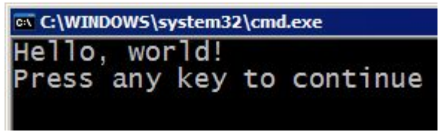
Programming basics

- **code** or **source code**: The sequence of instructions in a program.
- **syntax**: The set of legal structures and commands that can be used in a particular programming language.
- **output**: The messages printed to the user by a program.
- **console**: The text box onto which output is printed.
  - Some source code editors pop up the console as an external window, and others contain their own console window.



```
Python 2.5.1 (r251:54863, Apr 18 2
win32
Type "copyright", "credits" or "li

    ****************************
    Personal firewall software may
    makes to its subprocess using
    interface.  This connection is
    interface and no data is sent
    ****************************

IDLE 1.2.1
>>> ================================ RESTART ================================
>>>
Hello, world!
4
>>>
```

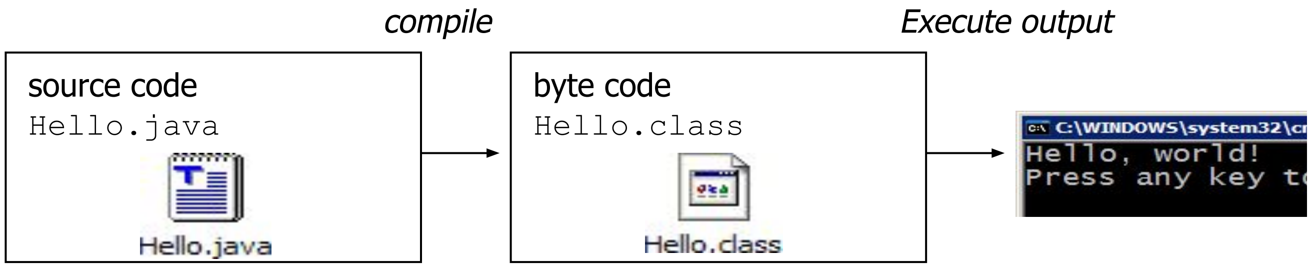
```
print "Hello, world!"
print 2 + 2
```



```
Hello, world!
Press any key to continue
```

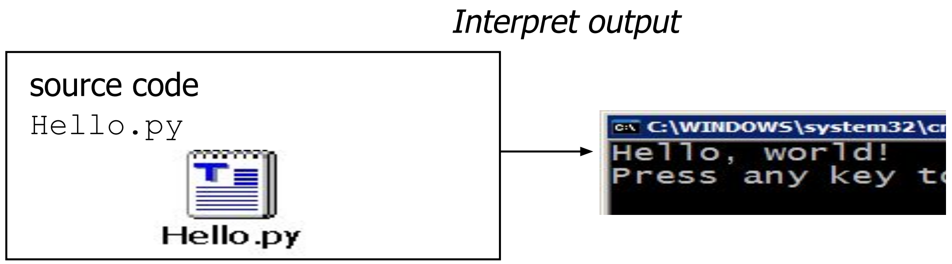
Compiling and interpreting

- **Java or C**



```
Hello, world!
Press any key tc
```

- **Python or R**

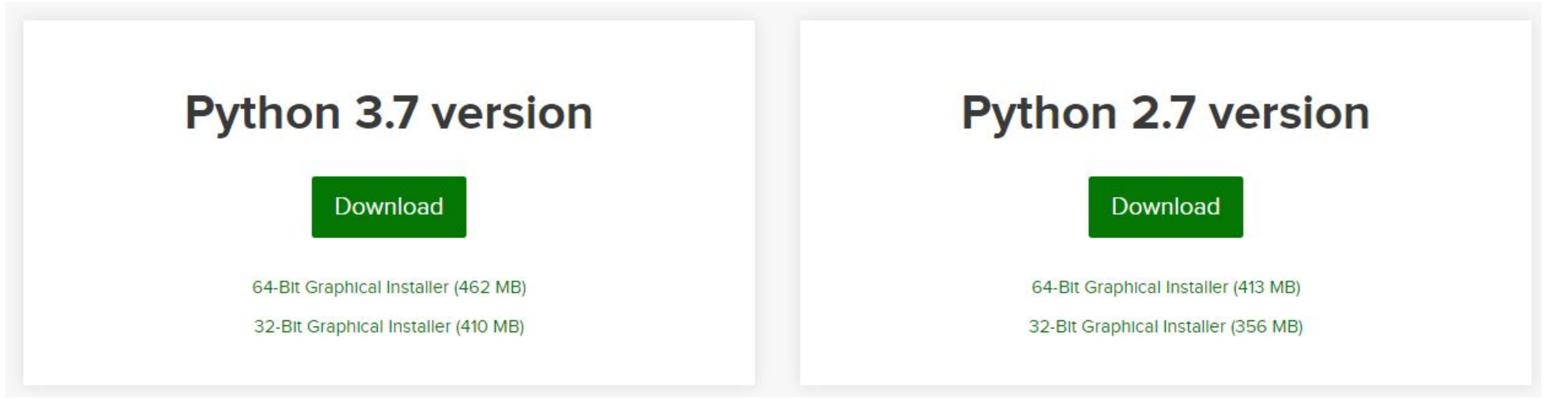


```
Hello, world!
Press any key tc
```

Installing Python with Anaconda

- Anaconda: free and open source platform for Python and R which embeds Python/R, several IDEs and packages for DataScience

- Download Anaconda here:
  https://www.anaconda.com/distribution/#download-section



Python 3.7 version

64-Bit Graphical Installer (462 MB)

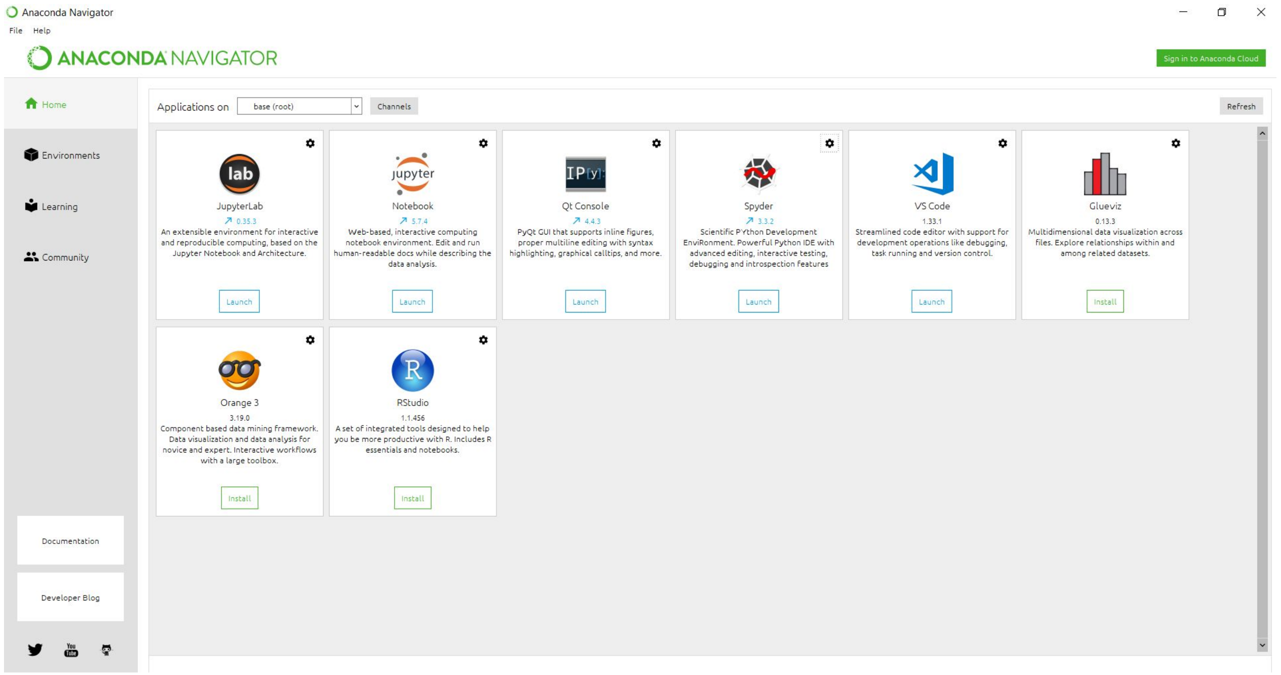
32-Bit Graphical Installer (410 MB)

Python 2.7 version

64-Bit Graphical Installer (413 MB)

32-Bit Graphical Installer (356 MB)

Installing Python with Anaconda



JupyterLab
0.35.3
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook
5.7.4
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Qt Console
4.4.3
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Spyder
3.3.2
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

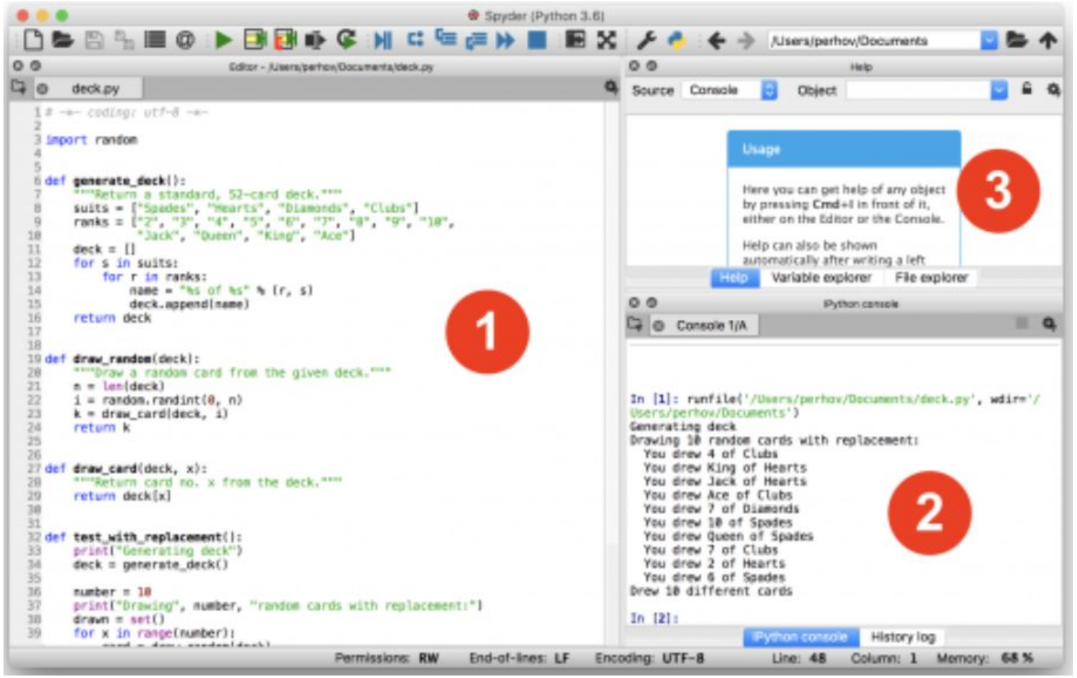
VS Code
1.33.1
Streamlined code editor with support for development operations like debugging, task running and version control.

Glueviz
0.13.3
Multidimensional data visualization across files. Explore relationships within and among related datasets.

Orange 3
3.19.0
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

RStudio
1.1.456
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

Spyder



```python
# -*- coding: utf-8 -*-

import random


def generate_deck():
    """Return a standard, 52-card deck."""
    suits = ["Spades", "Hearts", "Diamonds", "Clubs"]
    ranks = ["2", "3", "4", "5", "6", "7", "8", "9", "10",
             "Jack", "Queen", "King", "Ace"]
    deck = []
    for s in suits:
        for r in ranks:
            name = "%s of %s" % (r, s)
            deck.append(name)
    return deck


def draw_random(deck):
    """Draw a random card from the given deck."""
    n = len(deck)
    i = random.randint(0, n)
    k = draw_card(deck, i)
    return k


def draw_card(deck, x):
    """Return card no. x from the deck."""
    return deck[x]


def test_with_replacement():
    print("Generating deck")
    deck = generate_deck()

    number = 10
    print("Drawing", number, "random cards with replacement:")
    drawn = set()
    for x in range(number):
```

Here you can get help of any object by pressing Cmd+I in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left

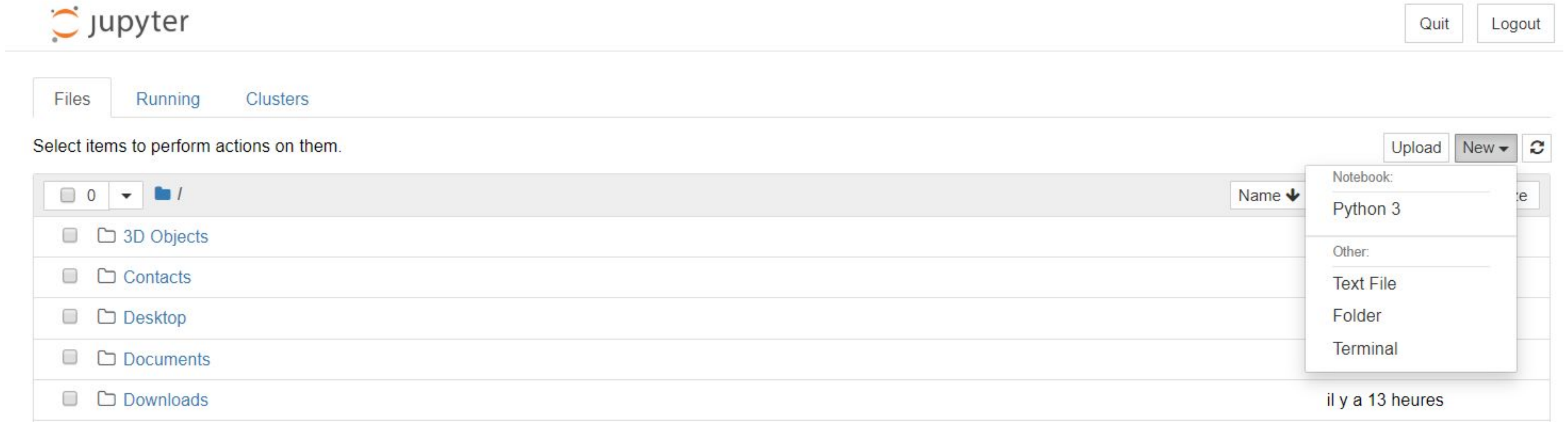
```
In [1]: runfile('/Users/perhov/Documents/deck.py', wdir='/
Users/perhov/Documents')
Generating deck
Drawing 10 random cards with replacement:
  You drew 4 of Clubs
  You drew King of Hearts
  You drew Jack of Hearts
  You drew Ace of Clubs
  You drew 7 of Diamonds
  You drew 10 of Spades
  You drew Queen of Spades
  You drew 7 of Clubs
  You drew 2 of Hearts
  You drew 6 of Spades
Drew 10 different cards

In [2]:
```

Permissions: RW   End-of-lines: LF   Encoding: UTF-8   Line: 48   Column: 1   Memory: 68 %

1. **The editor:** Where you edit the source code.
2. **The Python console:** Displays the program's output and lets you run python commands interactively.
3. **The help window / variable explorer / file explorer:** Use the buttons below the window to select which one to use.

Jupyter Notebooks

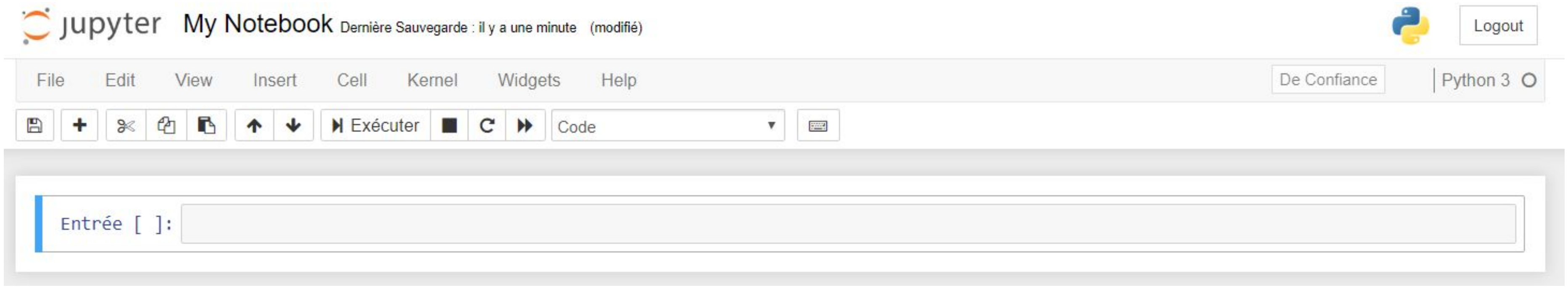
http://localhost:8888/tree



Select items to perform actions on them.

3D Objects
Contacts
Desktop
Documents
Downloads il y a 13 heures

Jupyter Notebooks



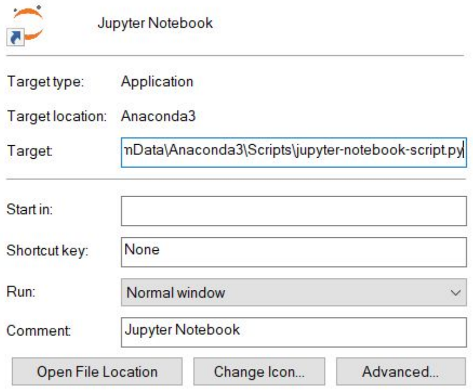
```
Entrée [ ]:
```

Change Jupyter Notebook Working directory

If you want to work with Jupyter notebook in a specific directory:

- Open a terminal
- Use the command line: jupyter notebook --generate-config
- Open the file: C:\Users\XXX\.jupyter\jupyter_notebook_config.py
- Change the notebook directory for: c.NotebookApp.notebook_dir = 'C:\\Users\\'

Jupyter will then by default open the directory: 'C:\\Users\\'

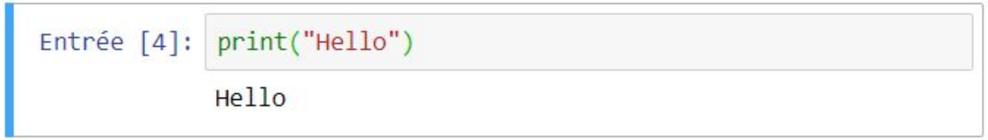


Jupyter Notebook

Target type:	Application
Target location:	Anaconda3
Target:	nData\Anaconda3\Scripts\jupyter-notebook-script.py
Start in:	
Shortcut key:	None
Run:	Normal window
Comment:	Jupyter Notebook

Run a first Python script in Jupyter

- Open a new Python3 notebook in Jupyter
- Create a new “cell” box
- Write the below and click on “Run” or Cell / Run all



```
Entrée [4]: print("Hello")

            Hello
```

- Click on Save (Upper left button)

4. Values and Variables

Python Objects

All the data in a Python code is represented by objects or by relations between objects. Every object has an identity, a type, and a value.

- Identity: never changes once it has been created, think of it as the object’s address in memory
- Type: An object’s type defines the possible values and operations that type supports. The type() function returns the type of an object. An object type is unchangeable like the identity
- Value: The value of some objects can change. Objects whose value can change are said to be mutable; objects whose value is unchangeable once they are created are called immutable.

Main Python Objects

Mutable :

List

Set

Dictionary

Immutable:

Int

Float

Bool

String

Tuple

Values and variables

Let's explore the various building blocks that are used to develop Python programs:

- Numeric values
- Strings
- Variables
- Assignments
- Identifiers
- Reserved words

Integers and String Values

The number four (5) is an example of a numeric value. In mathematics, 5 is an integer value. Integers are whole numbers, which means they have no fractional parts, and they can be positive, negative, or zero. Python supports a number of numeric and nonnumeric values. In particular, Python programs can use integer values. The Python statement:

```
print(5)
```

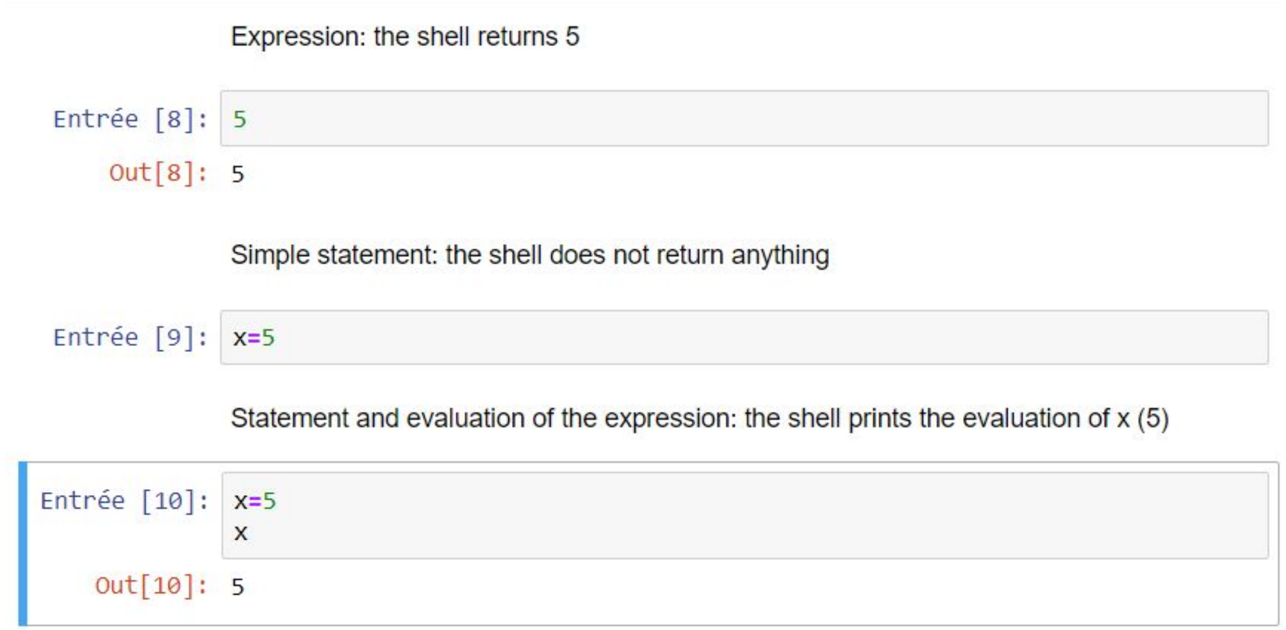
displays the value 5. Note that no quotation marks appear in the statement. The number 5 itself is not a complete python statement and is therefore not a program. However if you enter 5 directly into the interactive interpreter shell python will also return the value 5.

Integers and String Values

The interactive shell attempts to evaluate both expressions and statements. In this case, the expression 5 evaluates to 5. The shell executes what is commonly called the read, eval, print loop. This means the interactive shell’s activity consists of:

- reading the text entered by the user
- attempting to evaluate the user’s input in the context of what the user has entered up that point
- printing its evaluation of the user’s input.

Integers and String Values


Expression: the shell returns 5

```
Entrée [8]: 5
    Out[8]: 5
```

Simple statement: the shell does not return anything

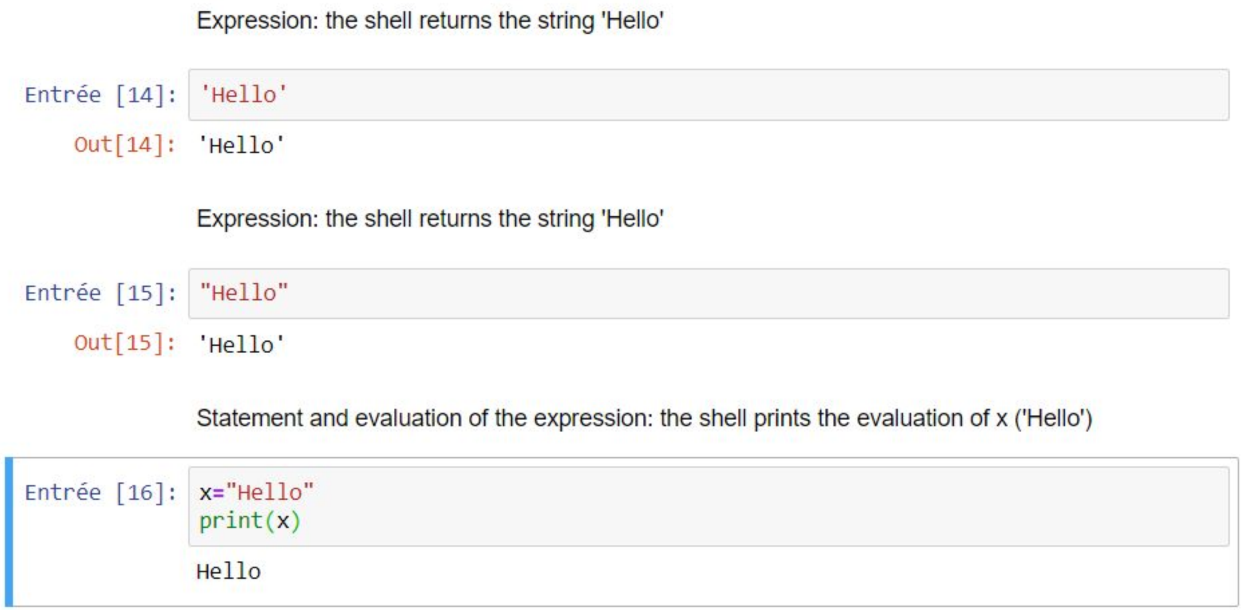
```
Entrée [9]: x=5
```

Statement and evaluation of the expression: the shell prints the evaluation of x (5)

```
Entrée [10]: x=5
             x
    Out[10]: 5
```

Integers and String Values

- Strings: Python supports both single and double quotes



Expression: the shell returns the string 'Hello'

```
Entrée [14]: 'Hello'

   Out[14]: 'Hello'
```

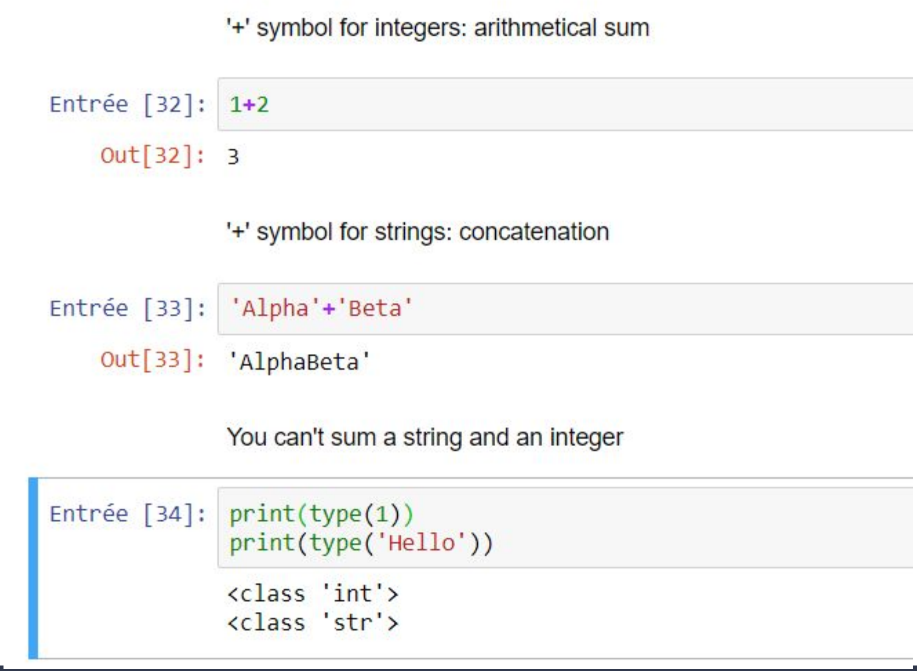
Expression: the shell returns the string 'Hello'

```
Entrée [15]: "Hello"

   Out[15]: 'Hello'
```

Statement and evaluation of the expression: the shell prints the evaluation of x ('Hello')

```
Entrée [16]: x="Hello"
             print(x)

             Hello
```

Integers and String Values: + symbol


'+' symbol for integers: arithmetical sum

```
Entrée [32]: 1+2
   Out[32]: 3
```

'+' symbol for strings: concatenation

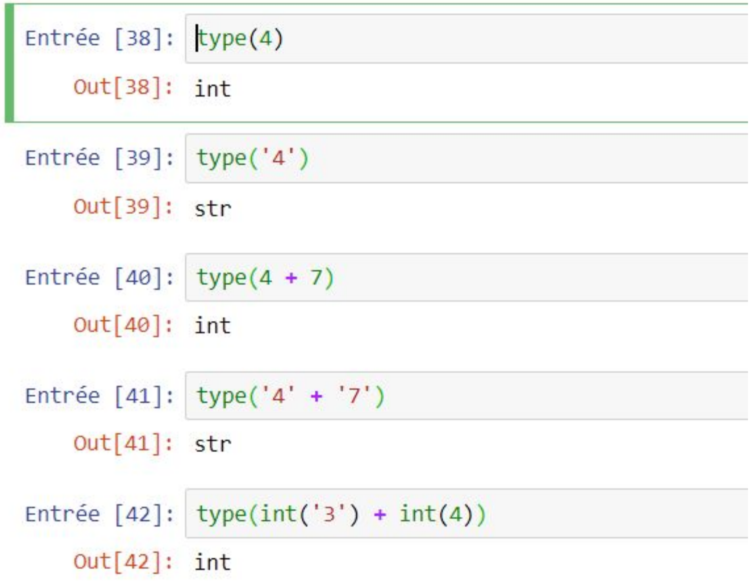
```
Entrée [33]: 'Alpha'+'Beta'
   Out[33]: 'AlphaBeta'
```

You can't sum a string and an integer

```
Entrée [34]: print(type(1))
             print(type('Hello'))

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

Integers and String Values:type function

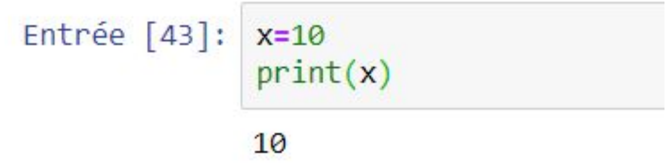
- The type function can determine the type of the most complicated expressions:



```
Entrée [38]: type(4)
    Out[38]: int

Entrée [39]: type('4')
    Out[39]: str

Entrée [40]: type(4 + 7)
    Out[40]: int

Entrée [41]: type('4' + '7')
    Out[41]: str

Entrée [42]: type(int('3') + int(4))
    Out[42]: int
```

Variables and assignments

- In python variables can represent values other than numbers. The below script uses a variable to store an integer value and then prints the value of the variable.

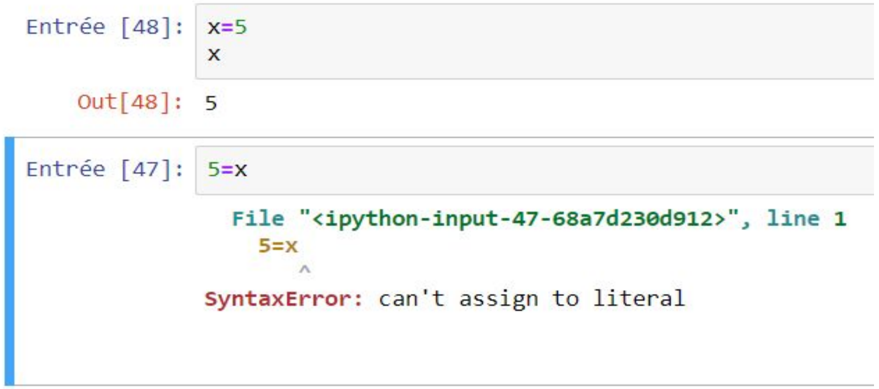


```
Entrée [43]: x=10
             print(x)

10
```

This is an assignment statement. An assignment statement associates a value with a variable. The key to an assignment statement is the symbol = which is known as the assignment operator. The statement assigns the integer value 10 to the variable x. Said another way, this statement binds the variable named x to the value 10. At this point the type of x is int because it is bound to an integer value.
We may assign and reassign a variable as often as necessary. The type of a variable will change if it is reassigned an expression of a different type.

Variables and assignments

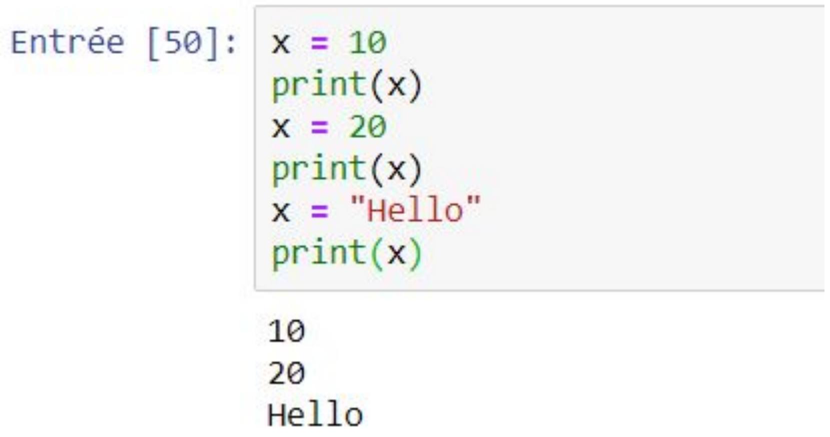
- = Symbol
  The meaning of the assignment operator (=) is different from equality in mathematics. In mathematics, = asserts that the expression on its left is equal to the expression on its right. In Python, = makes the variable on its left take on the value of the expression on its right. It is best to read x = 5 as “x is assigned the value 5.



```
Entrée [48]: x=5
             x

   Out[48]: 5

Entrée [47]: 5=x

  File "<ipython-input-47-68a7d230d912>", line 1
    5=x
       ^
SyntaxError: can't assign to literal
```

Variables and assignments

- Multiple assignments
  We can reassign different values to a variable as needed. Note that we don’t need to declare a new variable with a different type



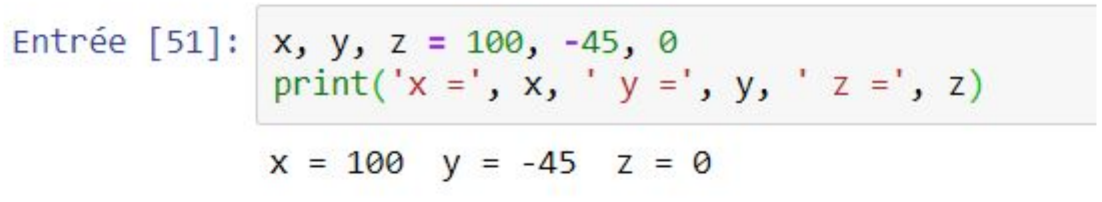
```
Entrée [50]: x = 10
             print(x)
             x = 20
             print(x)
             x = "Hello"
             print(x)
```

```
10
20
Hello
```

Variables and assignments

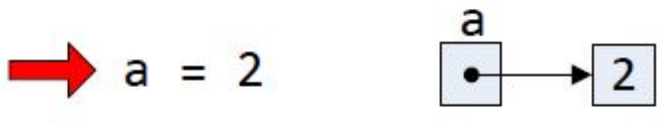
- Tuple assignments

A tuple is a comma-separated list of expressions. Tuple assignment works as follows: The first variable in the tuple on left side of the assignment operator is assigned the value of the first expression in the tuple on the left side (effectively x = 100). Similarly, the second variable in the tuple on left side of the assignment operator is assigned the value of the second expression in the tuple on the left side (in effect y = -45). z gets the value 0.

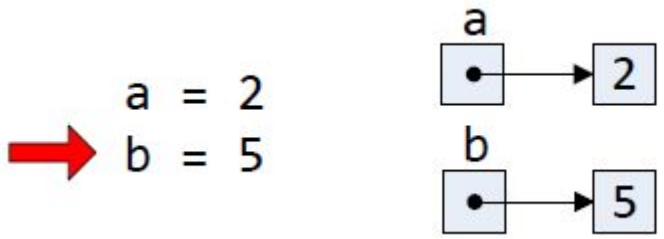

Entrée [51]:
```
x, y, z = 100, -45, 0
print('x =', x, ' y =', y, ' z =', z)
```
```
x = 100  y = -45  z = 0
```

Variables and assignments

- Re-assignments



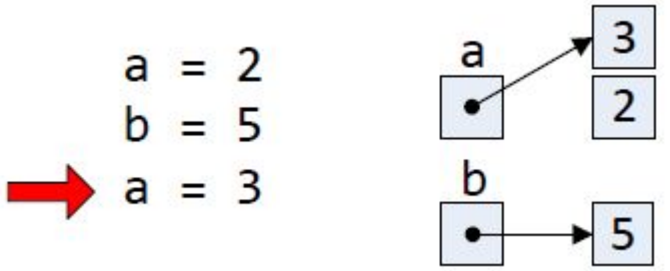
```
a = 2
```



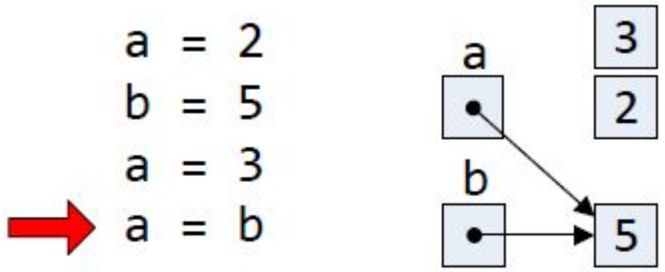
```
a = 2
b = 5
```

Variables and assignments

- Re-assignments



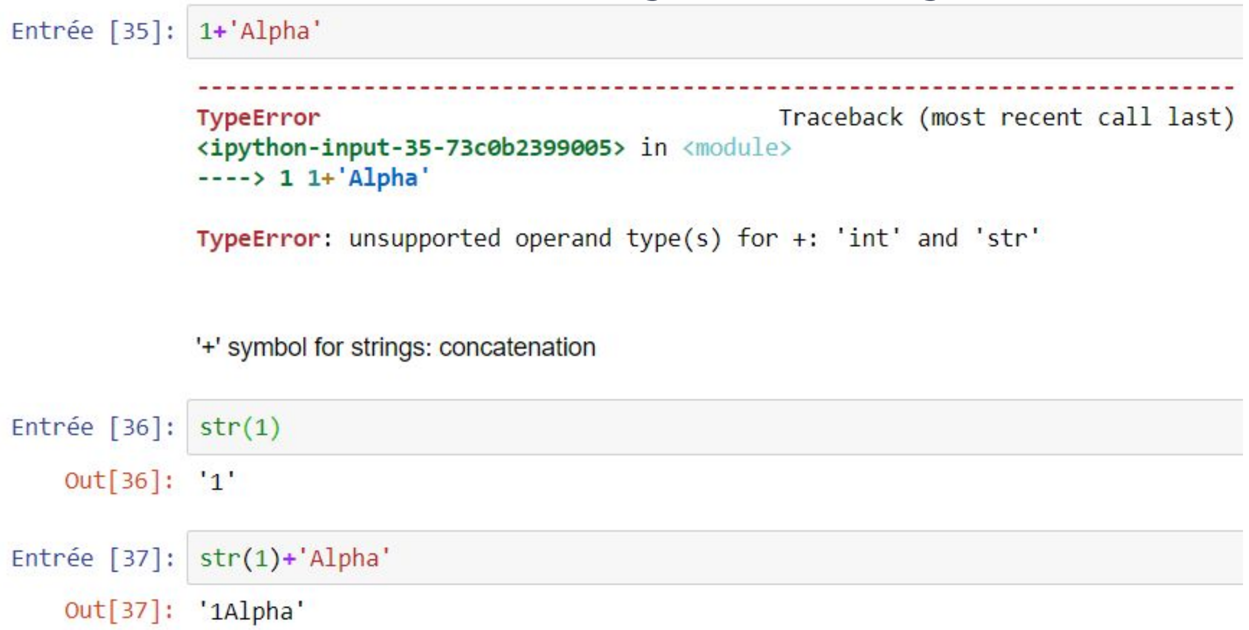
```
a = 2
b = 5
a = 3
```



```
a = 2
b = 5
a = 3
a = b
```

Integers and String Values: + symbol

- ‘Str’ function to convert an integer into a string



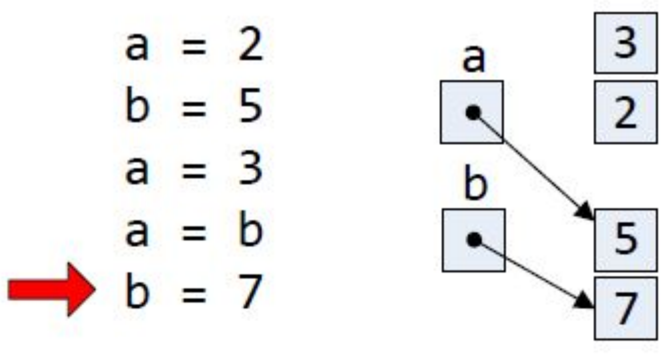
```
Entrée [35]: 1+'Alpha'

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-35-73c0b2399005> in <module>
----> 1 1+'Alpha'

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

'+' symbol for strings: concatenation

```
Entrée [36]: str(1)

   Out[36]: '1'

Entrée [37]: str(1)+'Alpha'

   Out[37]: '1Alpha'
```

Variables and assignments

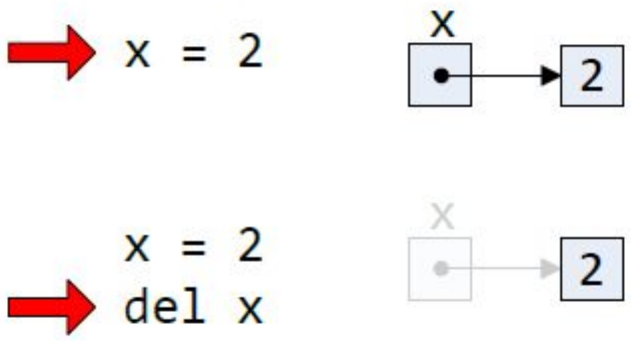
- Re-assignments

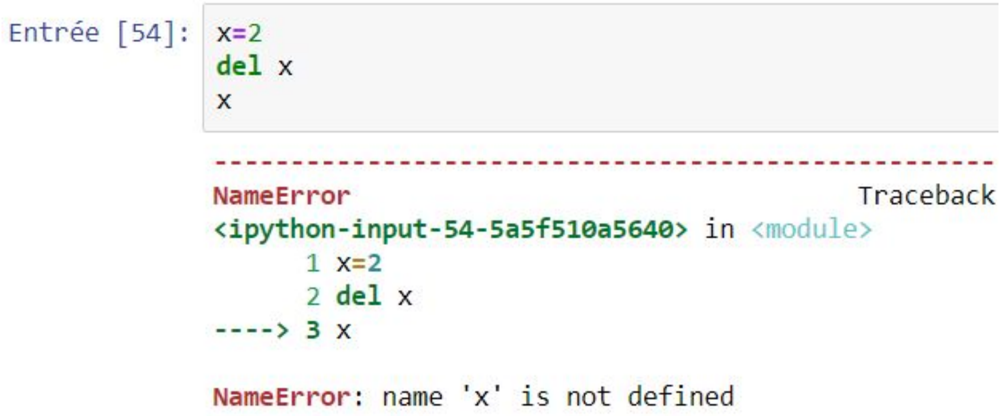


```
a = 2
b = 5
a = 3
a = b
b = 7
```

Variables and assignments

- del deletes or removes a variable's definition





```
Entrée [54]: x=2
             del x
             x
```

```
---------------------------------------------------------------------------
NameError                                 Traceback
<ipython-input-54-5a5f510a5640> in <module>
      1 x=2
      2 del x
----> 3 x

NameError: name 'x' is not defined
```

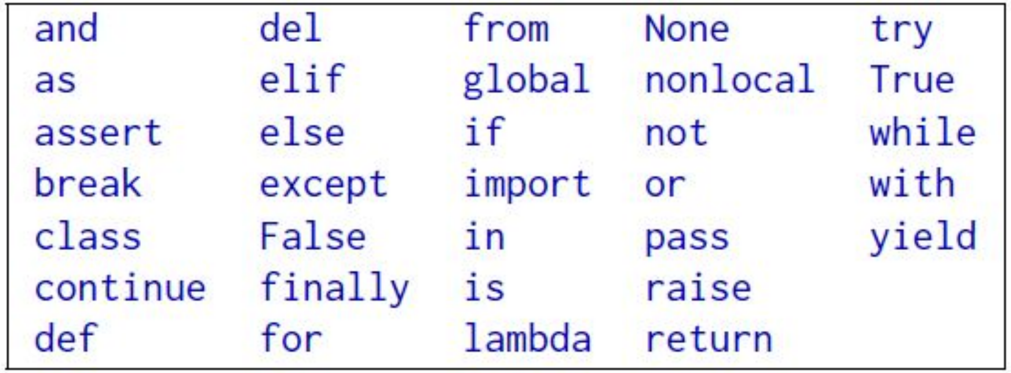
Identifiers / Variables names

An identifier is a word used to name things. A variable name is one example of an identifier. An identifier must contain at least one character. Python has strict rules for variable names:

- The first character of an identifiers must be an alphabetic letter or the underscore
  ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_
- The remaining characters (if any) may be alphabetic characters (upper or lower case), the underscore or a digit
  ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_0123456789
- No other characters (including spaces) are permitted in identifiers.
- A reserved word cannot be used as an identifier (see Table below).

Identifiers / Variables names

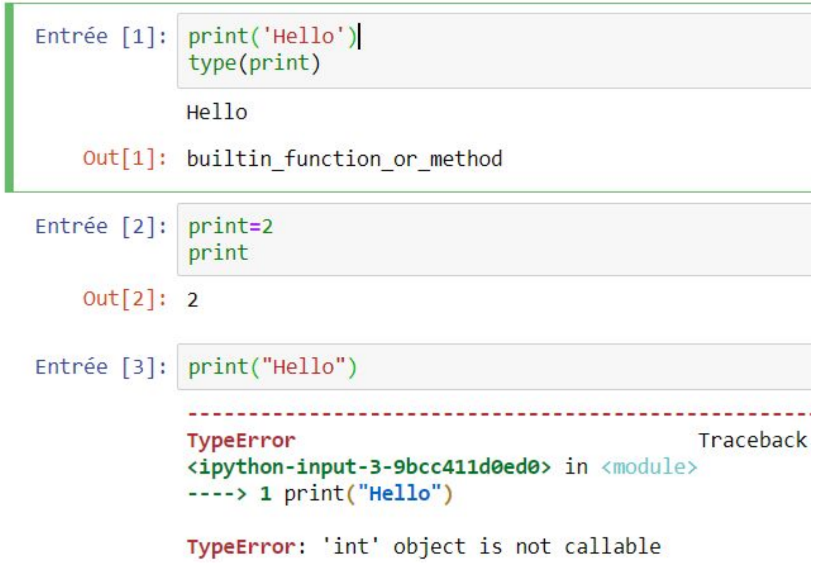
Python keywords (can’t be used as identifiers)



and	del	from	None	try
as	elif	global	nonlocal	True
assert	else	if	not	while
break	except	import	or	with
class	False	in	pass	yield
continue	finally	is	raise	
def	for	lambda	return	

Identifiers / Variables names

There is nothing special about the names print, int, str, or type, other than they happen to be the names of built-in functions. We are free to reassign these names:

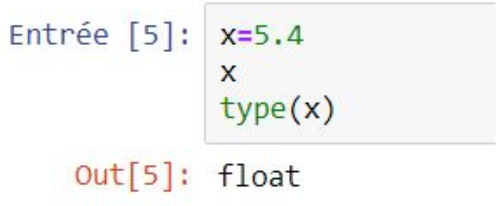


```
Entrée [1]: print('Hello')
            type(print)

            Hello

    Out[1]: builtin_function_or_method

Entrée [2]: print=2
            print

    Out[2]: 2

Entrée [3]: print("Hello")

            ---------------------------------------------------
            TypeError                                 Traceback
            <ipython-input-3-9bcc411d0ed0> in <module>
            ----> 1 print("Hello")

            TypeError: 'int' object is not callable
```

Here we used the name print as a variable. In so doing it lost its original behavior as a function to print the console. While we can reassign the names print, str, type, etc., it generally is not a good idea to do so.

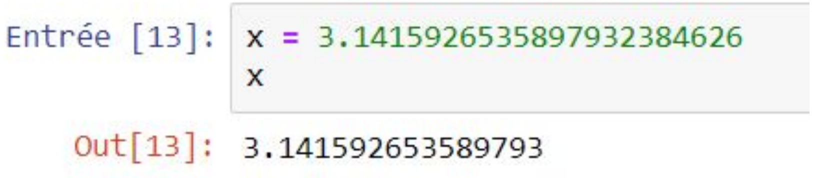
Floating point numbers

Many computational tasks require numbers that have fractional parts. Python supports such noninteger numbers, and they are called floating-point numbers.



```
Entrée [5]: x=5.4
            x
            type(x)

    Out[5]: float
```

Consider the real number pi. The mathematical constant p is an irrational number which means it contains an infinite number of digits with no pattern that repeats. Since p contains an infinite number of digits, a Python program can only approximate p's value

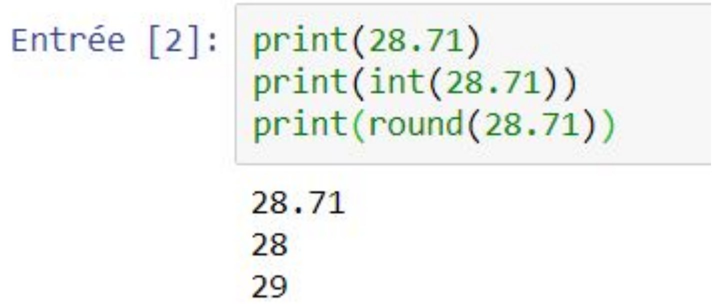


```
Entrée [13]: x = 3.14159265358979323846 26
             x

   Out[13]: 3.141592653589793
```

Floating point numbers

Unlike floating-point numbers, integers are whole numbers and cannot store fractional quantities. We can convert a floating-point to an integer in two fundamentally different ways:

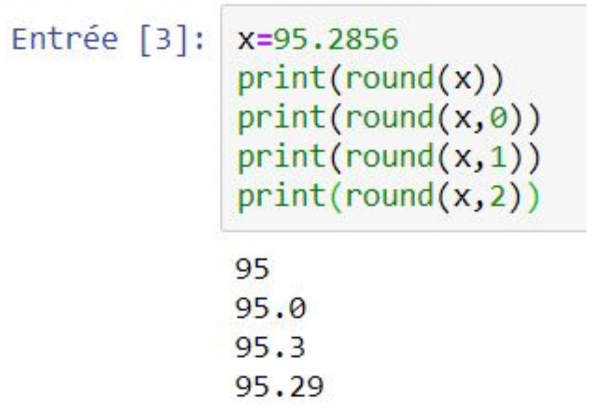
- **Rounding** adds or subtracts a fractional amount as necessary to produce the integer closest to the original floating-point value.
- **Truncation** simply drops the fractional part of the floating-point number, thus keeping whole number part that remains.



```
Entrée [2]: print(28.71)
            print(int(28.71))
            print(round(28.71))

28.71
28
29
```

Floating point numbers

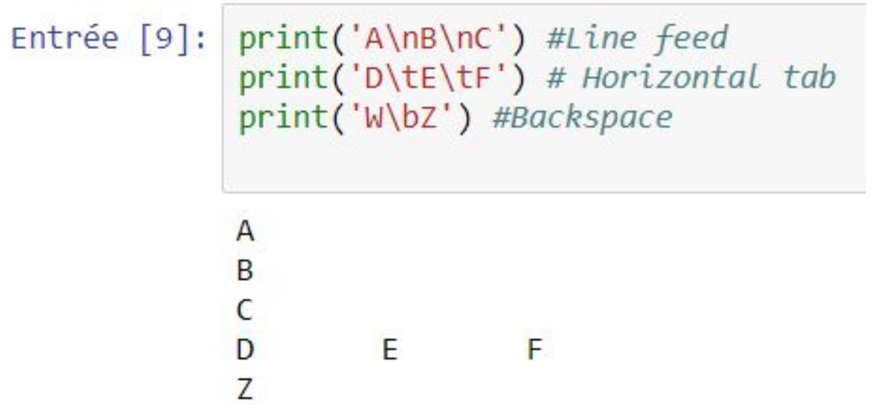
As we can see, truncation always “rounds down,” while rounding behaves as we would expect. We also can use the round function to round a floating-point number to a specified number of decimal places. The round function accepts an optional argument that produces a floating-point rounded to fewer decimal places. The additional argument must be an integer and specifies the desired number of decimal places to round



```
Entrée [3]: x=95.2856
            print(round(x))
            print(round(x,0))
            print(round(x,1))
            print(round(x,2))

            95
            95.0
            95.3
            95.29
```

Control codes within strings

The characters that can appear within strings include letters of the alphabet (A-Z, a-z), digits (0-9), punctuation (., :, ,, etc.), and other printable symbols (#, &, %, etc.). In addition to these “normal” characters, we may embed special characters known as **control codes**. Control codes control the way the console window or a printer renders text:

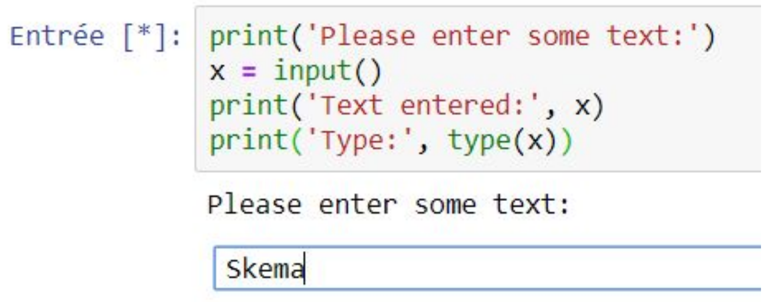


```
Entrée [9]: print('A\nB\nC') #Line feed
            print('D\tE\tF') # Horizontal tab
            print('W\bZ') #Backspace
```

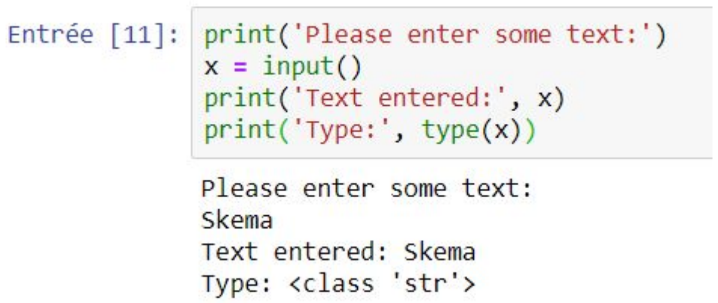
```
A
B
C
D       E       F
Z
```

User input

The print function enables a Python program to display textual information to the user. Programs may use the input function to obtain information from the user. The simplest use of the input function assigns a string to a variable:



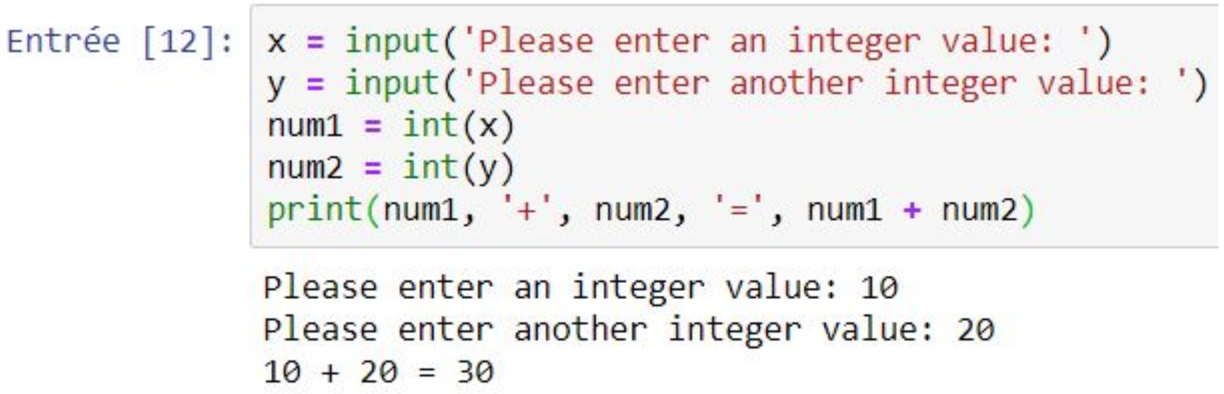
```
Entrée [*]: print('Please enter some text:')
            x = input()
            print('Text entered:', x)
            print('Type:', type(x))

            Please enter some text:

            Skema
```



```
Entrée [11]: print('Please enter some text:')
             x = input()
             print('Text entered:', x)
             print('Type:', type(x))

             Please enter some text:
             Skema
             Text entered: Skema
             Type: <class 'str'>
```

User input

Since user input almost always requires a message to the user about the expected input, the input function optionally accepts a string that it prints just before the program stops to wait for the user to respond. The statement x = input('Please enter some text: ') prints the message Please enter some text: and then waits to receive the user's input to assign to x.

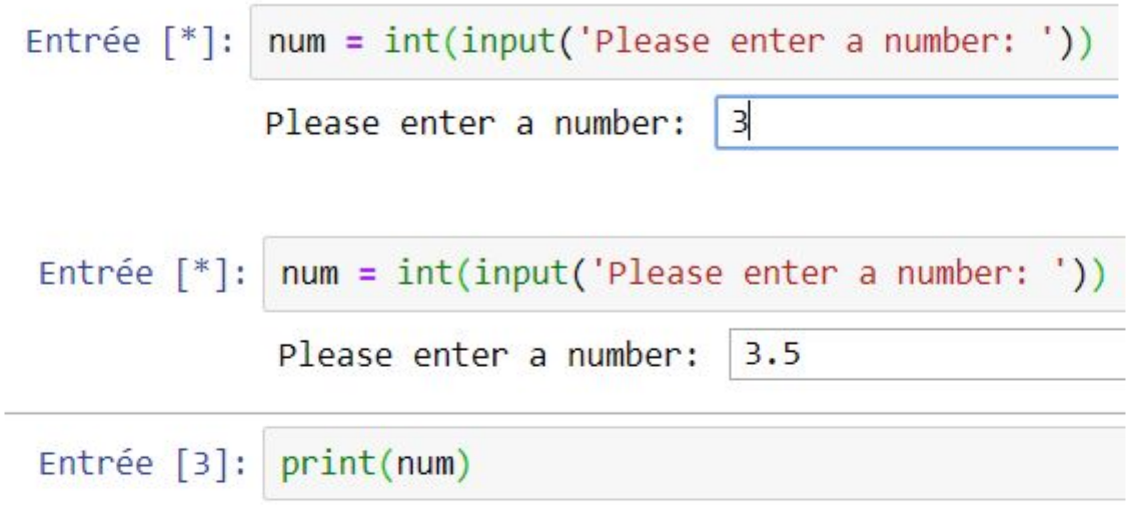


```
Entrée [12]: x = input('Please enter an integer value: ')
             y = input('Please enter another integer value: ')
             num1 = int(x)
             num2 = int(y)
             print(num1, '+', num2, '=', num1 + num2)
```

```
Please enter an integer value: 10
Please enter another integer value: 20
10 + 20 = 30
```

User input

Question: What are the outputs of the below python scripts ?



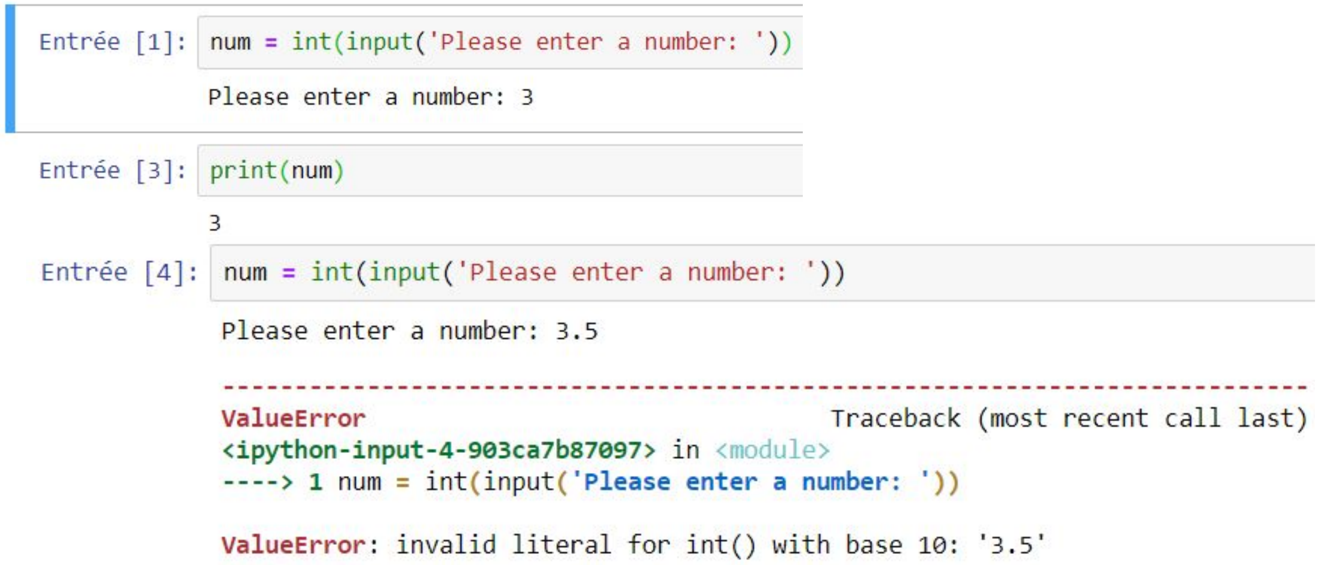
```
Entrée [*]: num = int(input('Please enter a number: '))

            Please enter a number: 3
```

```
Entrée [*]: num = int(input('Please enter a number: '))

            Please enter a number: 3.5
```

```
Entrée [3]: print(num)
```

User input

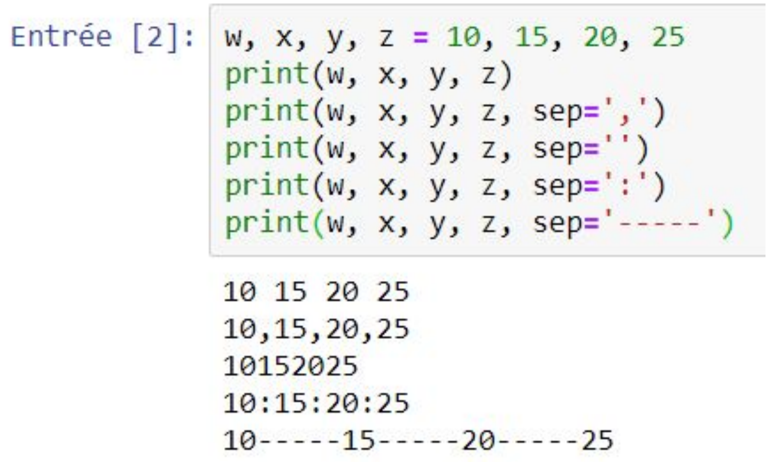
Anwer: first example works, second one returns an error. This example reveals that the int can convert a string to an integer only if the string looks exactly like an integer.



```
Entrée [1]: num = int(input('Please enter a number: '))

            Please enter a number: 3

Entrée [3]: print(num)

            3

Entrée [4]: num = int(input('Please enter a number: '))

            Please enter a number: 3.5

            ---------------------------------------------------------------------------
            ValueError                                Traceback (most recent call last)
            <ipython-input-4-903ca7b87097> in <module>
            ----> 1 num = int(input('Please enter a number: '))

            ValueError: invalid literal for int() with base 10: '3.5'
```

Controlling the Print function

A keyword argument allows us to control how the print function visually separates the arguments it displays. By default, the print function places a single space in between the items it prints. print uses a keyword argument named sep to specify the string to use insert between items. The name sep stands for separator. The default value of sep is the string ' ', a string containing a single space

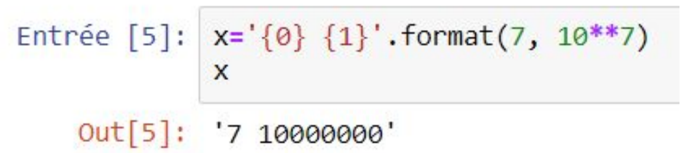


```
Entrée [2]: w, x, y, z = 10, 15, 20, 25
            print(w, x, y, z)
            print(w, x, y, z, sep=',')
            print(w, x, y, z, sep='')
            print(w, x, y, z, sep=':')
            print(w, x, y, z, sep='-----')
```

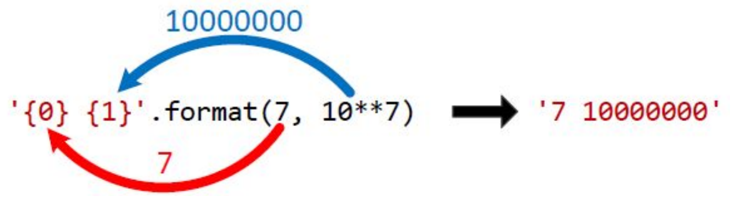
```
10 15 20 25
10,15,20,25
10152025
10:15:20:25
10-----15-----20-----25
```

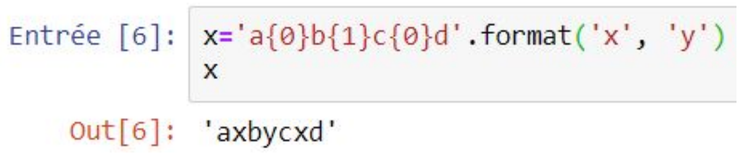
String formatting

You may need to print information combining variables and formatting string. Consider the below expression:

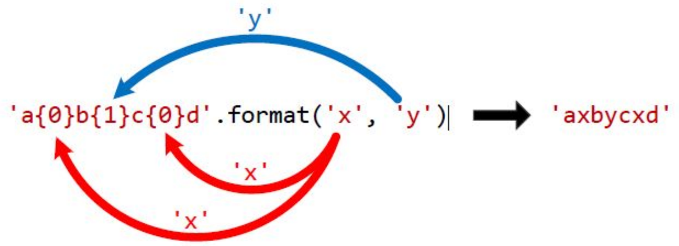


```
Entrée [5]: x='{0} {1}'.format(7, 10**7)
            x

   Out[5]: '7 10000000'
```



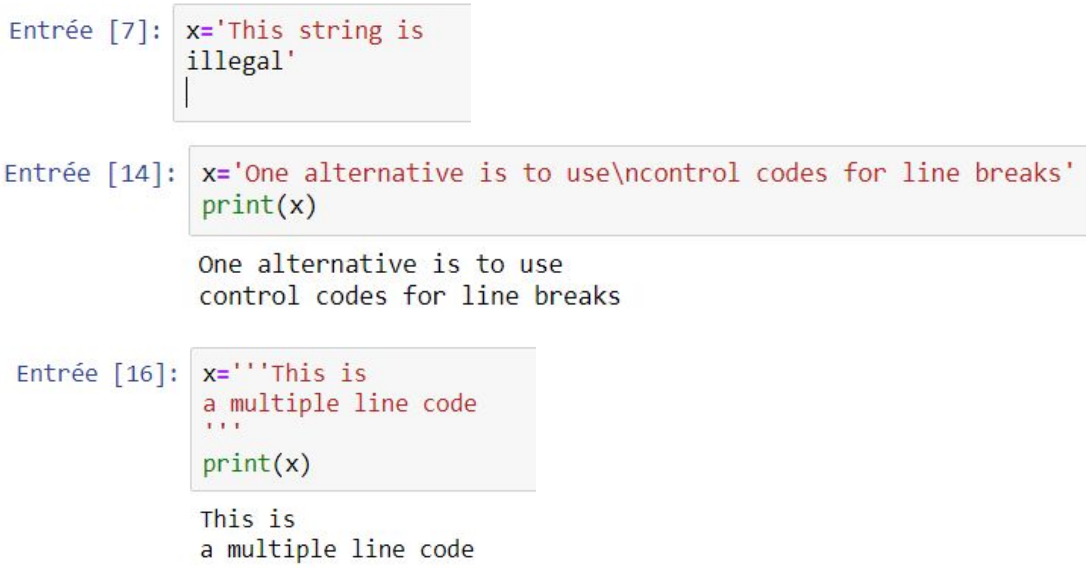


```
Entrée [6]: x='a{0}b{1}c{0}d'.format('x', 'y')
            x

   Out[6]: 'axbycxd'
```



Multi Lines strings

Multiple lines strings can be used using either control codes or triple quotes as below:



```
Entrée [7]:  x='This string is
             illegal'

Entrée [14]: x='One alternative is to use\ncontrol codes for line breaks'
             print(x)

             One alternative is to use
             control codes for line breaks

Entrée [16]: x='''This is
             a multiple line code
             '''
             print(x)

             This is
             a multiple line code
```

Exercises – Values and Variables

1/ Will the following lines of code print the same thing? Explain why or why not ?

```
x = 6
print(6)
print("6")
```

2/ Will the following lines of code print the same thing? Explain why or why not.

```
x = 7
print(x)
print("x")
```

3/ Once a variable has been properly assigned can its value be changed?

4/ In Python can you assign more than one variable in a single statement?

5/ Write a Python program containing exactly one print statement that produces the following output:

```
A
B
C
```

6/ Write a python program which ask the user to enter a word and then prints the word adding “ing” a the end

5. Expressions and Arithmetic

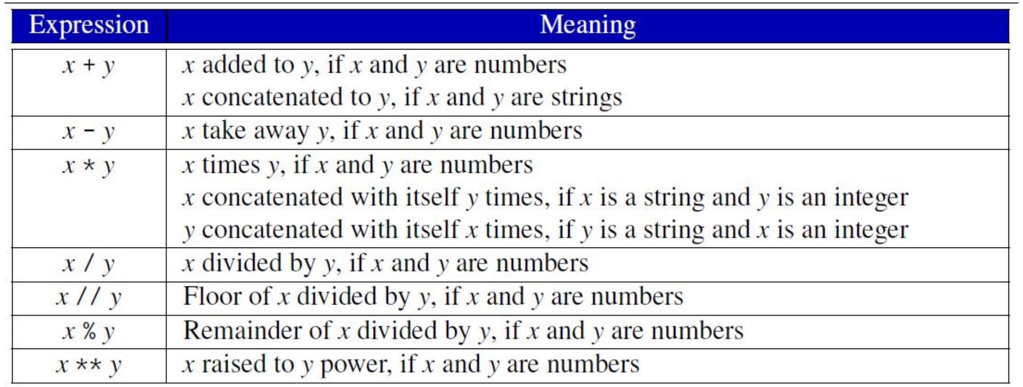
Expressions

- Expression: A data value or set of operations to compute a value.
  Examples: 1 + 4 * 3
- Precedence: Order in which operations are computed.

      * / % ** have a higher precedence than + -

      1 + 3 * 4 is 13

  Parentheses can be used to force a certain order of evaluation.



Expression	Meaning
$x + y$	x added to y, if x and y are numbers x concatenated to y, if x and y are strings
$x - y$	x take away y, if x and y are numbers
$x * y$	x times y, if x and y are numbers x concatenated with itself y times, if x is a string and y is an integer y concatenated with itself x times, if y is a string and x is an integer
x / y	x divided by y, if x and y are numbers
$x // y$	Floor of x divided by y, if x and y are numbers
$x \% y$	Remainder of x divided by y, if x and y are numbers
$x ** y$	x raised to y power, if x and y are numbers

Mixed Type Expressions

Expressions may contain mixed integer and floating-point elements; for example, in the following program Fragment:

```
x = 4
y = 10.2
sum = x + y
```

x is an integer and y is a floating-point number. What type is the expression x + y ? Except in the case of the / operator, arithmetic expressions that involve only integers produce an integer result. All arithmetic operators applied to floating-point numbers produce a floating-point result. When an operator has mixed operands—one operand an integer and the other a floating-point number the interpreter treats the integer operand as floating-point number and performs floating-point arithmetic. This means x + y is a floating point expression, and the assignment will make the variable sum bind to a floating-point value.

Math commands

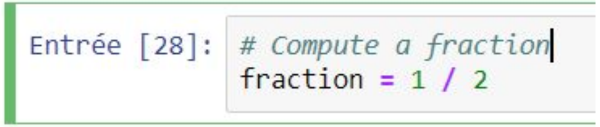
Command name	Description
`abs(`***value***`)`	absolute value
`ceil(`***value***`)`	rounds up
`cos(`***value***`)`	cosine, in radians
`floor(`***value***`)`	rounds down
`log(`***value***`)`	logarithm, base *e*
`log10(`***value***`)`	logarithm, base 10
`max(`***value1***`,` ***value2***`)`	larger of two values
`min(`***value1***`,` ***value2***`)`	smaller of two values
`round(`***value***`)`	nearest whole number
`sin(`***value***`)`	sine, in radians
`sqrt(`***value***`)`	square root

Constant	Description
`e`	2.7182818...
`pi`	3.1415926...

To use many of these commands, you must write the following at the top of your Python program:
from math import *

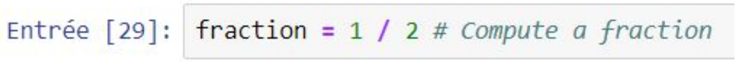
Comments

Any text contained within comments is ignored by the Python interpreter. The # symbol begins a comment in the source code. The comment is in effect until the end of the line of code:



```
Entrée [28]: # Compute a fraction
             fraction = 1 / 2
```

The first line here is a comment that explains what the statement that follows it is supposed to do. The comment begins with the # symbol and continues until the end of that line. The interpreter will ignore the # symbol and the contents of the rest of the line. You also may append a short comment to the end of a statement:



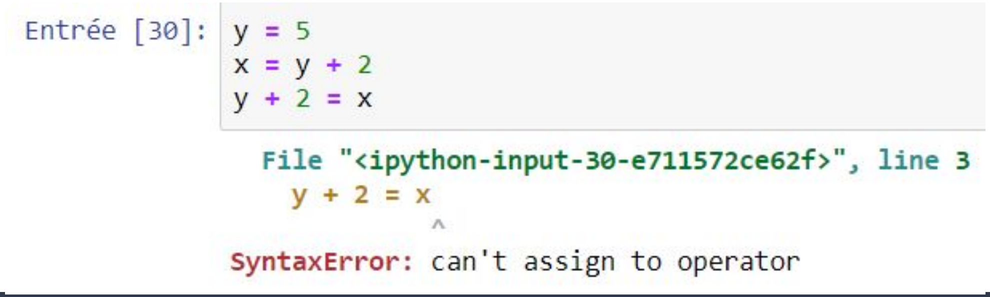
```
Entrée [29]: fraction = 1 / 2 # Compute a fraction
```

Errors

In Python, there are three general kinds of errors: syntax errors, run-time exceptions, and logic errors.

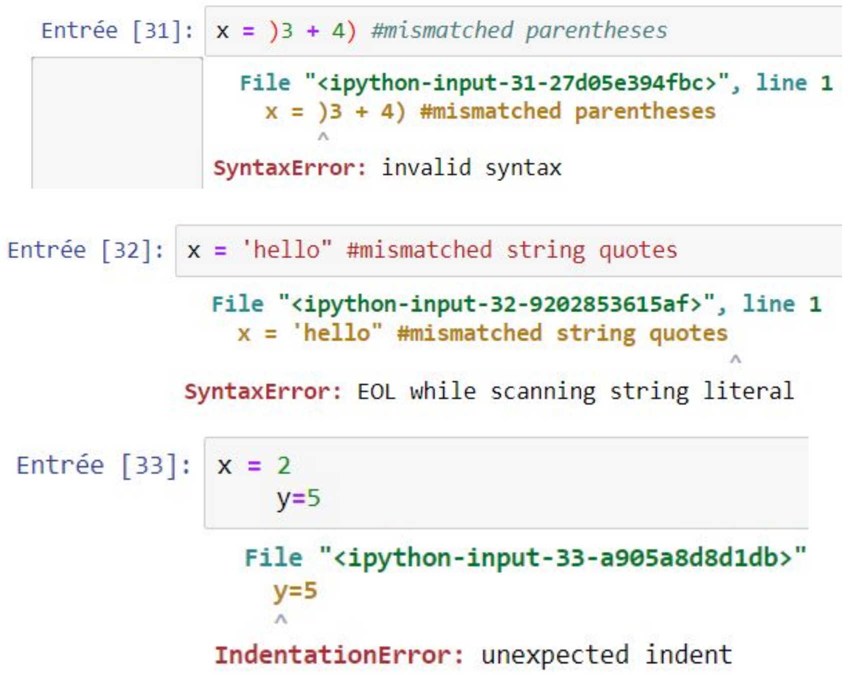
- Syntax errors:

If the interpreter detects an invalid program statement during the translation phase, it will terminate the program's execution and report an error. Such errors result from the programmer's misuse of the language



```
Entrée [30]: y = 5
             x = y + 2
             y + 2 = x

  File "<ipython-input-30-e711572ce62f>", line 3
    y + 2 = x
         ^
SyntaxError: can't assign to operator
```

Errors



```
Entrée [31]: x = )3 + 4) #mismatched parentheses

  File "<ipython-input-31-27d05e394fbc>", line 1
    x = )3 + 4) #mismatched parentheses
        ^
SyntaxError: invalid syntax
```

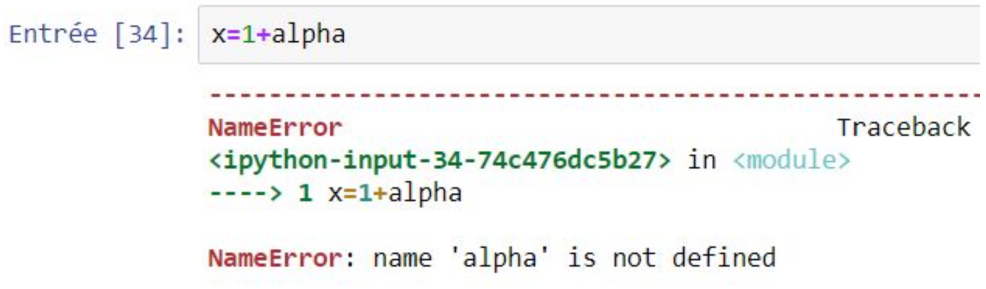
```
Entrée [32]: x = 'hello" #mismatched string quotes

  File "<ipython-input-32-9202853615af>", line 1
    x = 'hello" #mismatched string quotes
                                          ^
SyntaxError: EOL while scanning string literal
```

```
Entrée [33]: x = 2
                 y=5

  File "<ipython-input-33-a905a8d8d1db>"
    y=5
    ^
IndentationError: unexpected indent
```

Errors

- Run-Time exceptions:

A syntactically correct Python program still can have problems. Some language errors depend on the context of the program's execution. Such errors are called run-time exceptions or run-time errors

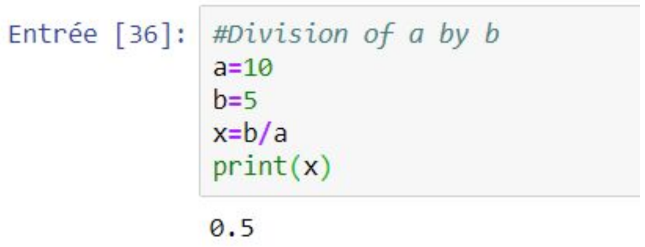


```
Entrée [34]: x=1+alpha

---------------------------------------------------------------------------
NameError                                 Traceback
<ipython-input-34-74c476dc5b27> in <module>
----> 1 x=1+alpha

NameError: name 'alpha' is not defined
```

Errors

- Logic errors:

The program contains an error, but the interpreter is unable detect the problem. It runs without the interpreter reporting any errors, but it produces incorrect results. An error of this type is known as a logic error.



```
Entrée [36]: #Division of a by b
             a=10
             b=5
             x=b/a
             print(x)

             0.5
```

More Arithmetic Operators

The statement: x = x + 1

increments x by one, making it one bigger than it was before this statement was executed. Python has a shorter statement that accomplishes the same effect: x += 1

More generally, you can use any statement of the form: x op = exp
Where:

- x is a variable.
- op= is an arithmetic operator combined with the assignment operator; for our purposes, the ones most useful to us are +=, -=, *=, /=, //=, and %=.
- exp is an expression compatible with the variable x.

Example: x *= y + z is equivalent to: x = x * (y + z)

Exercises: Expressions and Arithmetic

1/ Consider the formula to :convert temperature from degrees Fahrenheit to degrees Celsius:

C =5/9 x (F - 32)

Write a python program which request the user to enter e temperature in Fahrenheit and return a temperature in Celsius

2/ Using integer division and modulus, split up a given number of seconds to hours, minutes, and seconds

Exercises: Expressions and Arithmetic

3/ Given the following assignment: x = 2

Indicate what each of the following Python statements would print.
(a) print("x")
(b) print('x')
(c) print(x)
(d) print("x + 1")
(e) print('x' + 1)
(f) print(x + 1)

Exercises: Expressions and Arithmetic

4/ Consider the following program that attempts to compute the circumference of a circle given the radius entered by the user. Given a circle's radius, r, the circle's circumference, C is given by the Formula: C = 2 Pi r

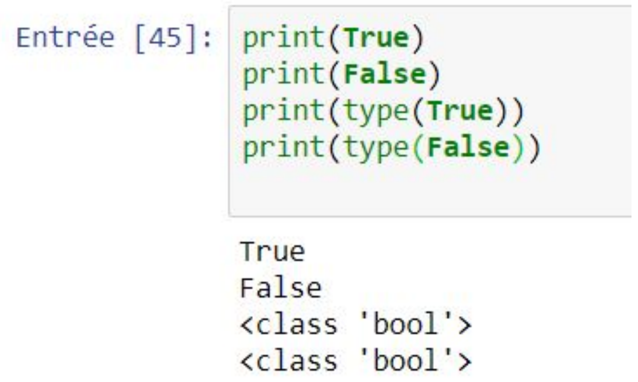
```
r = 0
PI = 3.14159
# Formula for the area of a circle given its radius
C = 2*PI*r
# Get the radius from the user
r = float(input("Please enter the circle's radius: "))
# Print the circumference
print("Circumference is", C)
```

(a) The program does not produce the intended result. Why?
(b) How can it be repaired so that it works correctly?

6. Conditional execution

Boolean expressions

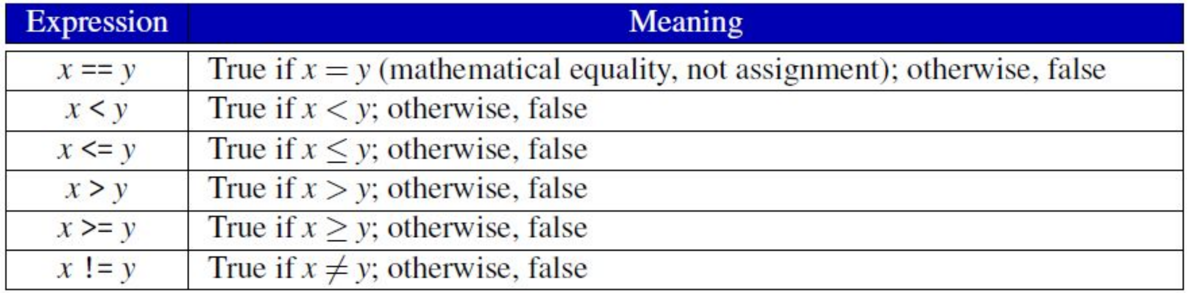
Arithmetic expressions evaluate to numeric values; a Boolean expression, sometimes called a predicate, may have only one of two possible values: false or true


Entrée [45]:
```
print(True)
print(False)
print(type(True))
print(type(False))
```

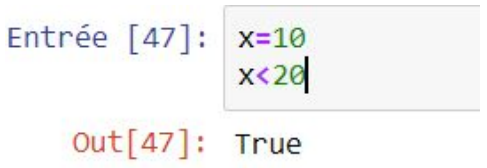
```
True
False
<class 'bool'>
<class 'bool'>
```

Boolean expressions

Python relational operators



Expression	Meaning
$x == y$	True if $x = y$ (mathematical equality, not assignment); otherwise, false
$x < y$	True if $x < y$; otherwise, false
$x <= y$	True if $x \leq y$; otherwise, false
$x > y$	True if $x > y$; otherwise, false
$x >= y$	True if $x \geq y$; otherwise, false
x != y	True if $x \neq y$; otherwise, false



```
Entrée [47]: x=10
             x<20

   Out[47]: True
```

Boolean expressions

The following expressions are all equivalent:

- x < 10
- 10 > x
- !(x >= 10)
- !(10 <= x)

IF Statement

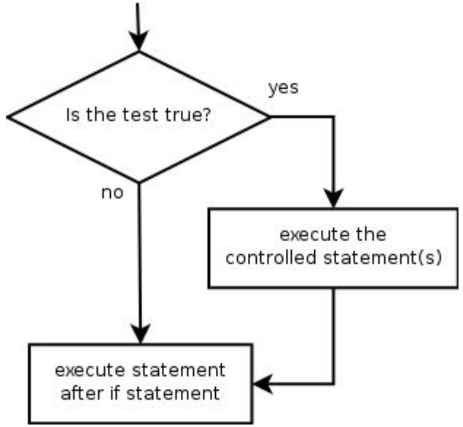
if statement: Executes a group of statements only if a certain condition is true. Otherwise, the statements are skipped.

Syntax:

```
if condition:
    statements
```

Example:

```
gpa = 3.4
if gpa > 2.0:
    print "Your application is accepted."
```


Is the test true?
yes
no
execute the controlled statement(s)
execute statement after if statement

IF Statement

if must be indented more spaces than the line that begins the if statement. The block technically is part of the if statement. This part of the if statement is sometimes called the body of the if. Python requires the block to be indented. If the block contains just one statement, some programmers will place it on the same line as the if; for example:

```
if x < 10:
        y = x
```

could be written:

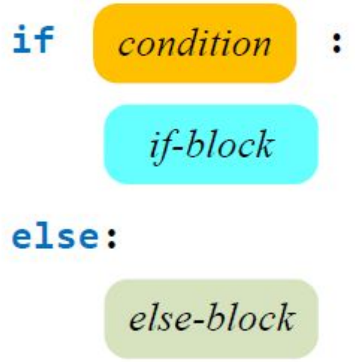
```
if x < 10: y = x
```

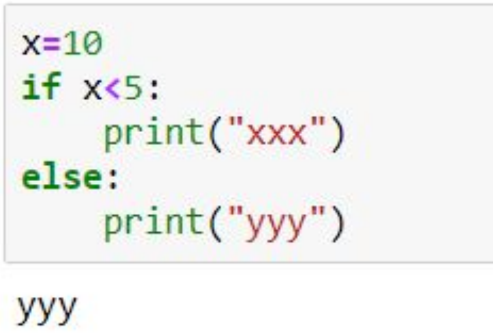
but may not be written as:

```
if x < 10:
y = x
```

ELSE and PASS Statements

The if statement has an optional else block that is executed only if the Boolean condition is false



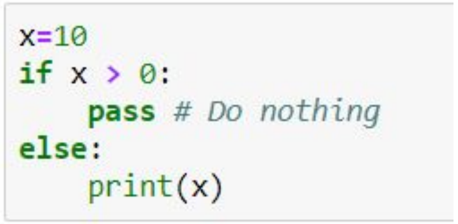


```
x=10
if x<5:
    print("xxx")
else:
    print("yyy")
```

```
yyy
```

ELSE and PASS Statements

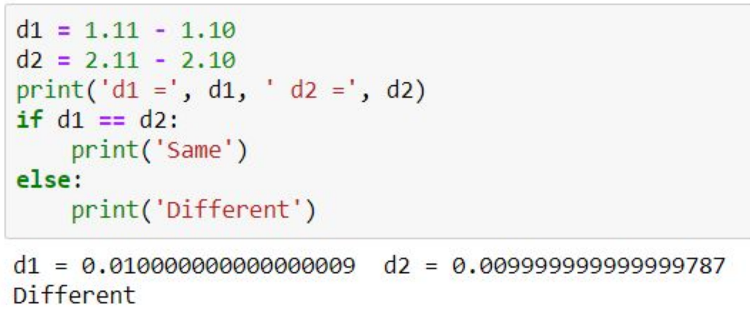
Python has a special statement, pass, that means do nothing. We may use the pass statement in our code in places where the language requires a statement to appear but we wish the program to take no action whatsoever. We can make the above code fragment legal by adding a pass statement:



```
x=10
if x > 0:
    pass # Do nothing
else:
    print(x)
```

Floating points equality

The equality operator (==) checks for exact equality. This can be a problem with floating-point numbers, since floating-point numbers inherently are imprecise. The below code demonstrates the perils of using the equality operator with floating-point numbers.



```
d1 = 1.11 - 1.10
d2 = 2.11 - 2.10
print('d1 =', d1, ' d2 =', d2)
if d1 == d2:
    print('Same')
else:
    print('Different')
```

```
d1 = 0.010000000000000009  d2 = 0.009999999999999787
Different
```

The solution is not to check floating-point numbers for exact equality, but rather see if the values “close enough” to each other to be considered the same

Logic complexity

Python provides the tools to construct some complicated conditional statements. Composing Boolean expressions with **and**, **or**, and **not** allows us to build conditions with arbitrarily complex logic. There are often are many ways to achieve the same effect; for example, the following four Boolean expressions are equivalent:

```
not (a == b and c != d)
not (a == b and not (c == d))
not (a == b) or not (c != d)
a != b or c == d
```

The solution is not to check floating-point numbers for exact equality, but rather see if the values "close enough" to each other to be considered the same

Exercises: Conditional Execution

1/ Using a “if” statement, write a python code which prints the absolute value of a number

2/ Write a Python program that requests an integer value from the user. If the value is between 1 and 100 inclusive, print ”OK;” otherwise, do not print anything.

3/ Using a “if” statement and logic, write a code which computes the maximum value of 3 integers entered by the user

4/ Write a Python program that requests 3 integer values from the user. It then prints one of two things: if any of the values entered are duplicates, it prints "DUPLICATES"; otherwise, it prints "ALL UNIQUE".

7. Iteration

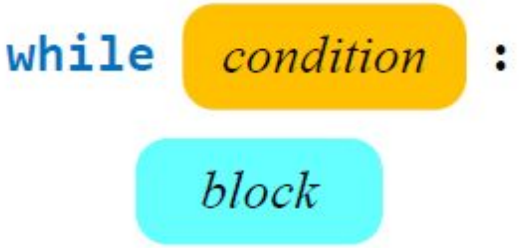
Iteration

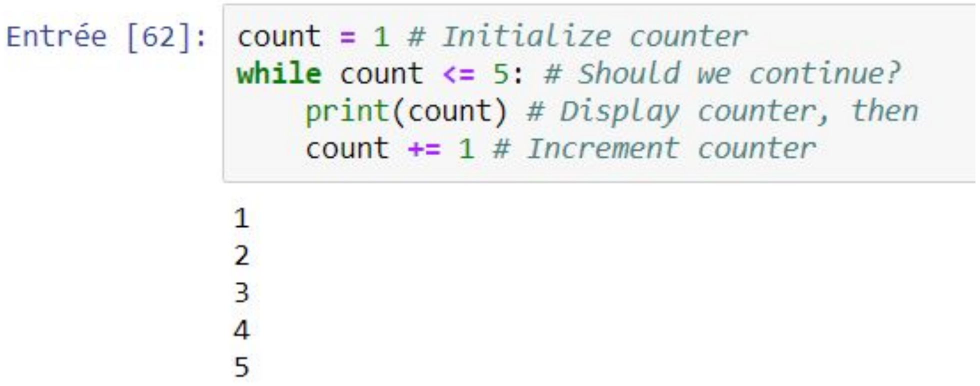
Iteration repeats the execution of a sequence of code. Iteration is useful for solving many programming problems. Iteration and conditional execution form the basis for algorithm construction.

The process of executing the same section of code over and over is known as iteration, or looping. Python has two different statements, **while** and **for**, that enable iteration.

The While statement

The while statement is ideal for indefinite loops.





```
Entrée [62]: count = 1 # Initialize counter
             while count <= 5: # Should we continue?
                 print(count) # Display counter, then
                 count += 1 # Increment counter
```

```
1
2
3
4
5
```

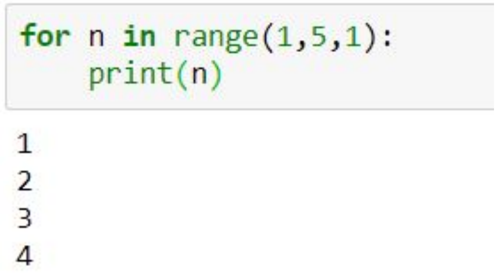
The For statement

Python provides a more convenient way to express a definite loop. The for statement iterates over a sequence of values. One way to express a sequence is via a tuple, as shown here:

```
for n in 1, 2, 3, 4, 5, 6, 7, 8, 9, 10:
        print(n)
```

This code behaves identically to the while loop above. The print statement here executes exactly 10 times. The code first prints 1, then 2, then 3, etc.

The For statement



```python
for n in range(1,5,1):
    print(n)
```

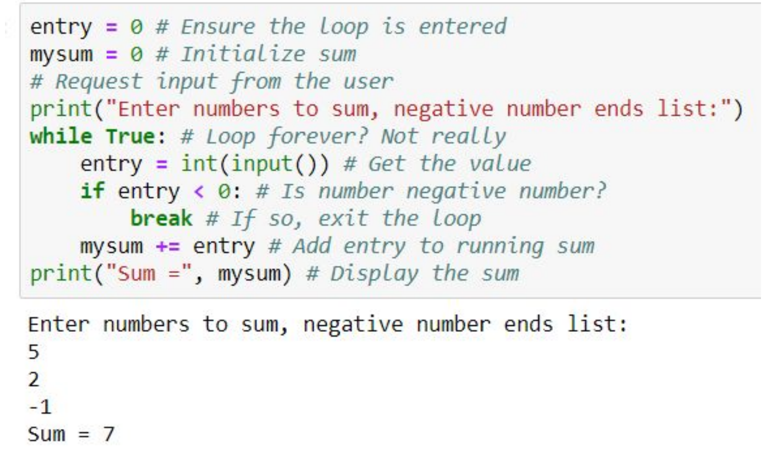
```
1
2
3
4
```

The general form of the **range** expression is : range(begin,end,step) where:

- begin is the first value in the range; if omitted, the default value is 0
- end is one past the last value in the range; the end value is always required and may not be omitted
- step is the amount to increment or decrement; if the step parameter is omitted, it defaults to 1 (counts up by ones)

The Break statement

Sometimes it is necessary to exit a loop from the middle of its body; that is, quit the loop before all the statements in its body execute. This means if a certain condition becomes true in the loop's body, exit right away. Python provides the break statement to implement middle-exiting loop control logic



```python
entry = 0 # Ensure the loop is entered
mysum = 0 # Initialize sum
# Request input from the user
print("Enter numbers to sum, negative number ends list:")
while True: # Loop forever? Not really
    entry = int(input()) # Get the value
    if entry < 0: # Is number negative number?
        break # If so, exit the loop
    mysum += entry # Add entry to running sum
print("Sum =", mysum) # Display the sum
```

```
Enter numbers to sum, negative number ends list:
5
2
-1
Sum = 7
```

Exercises: iteration

1/ Using a “for” and a “range” statement, write a python program which computes the sum of the squares from 1 to 10

2/ Using a “while” statement, write a python program which estimates the square root of a number

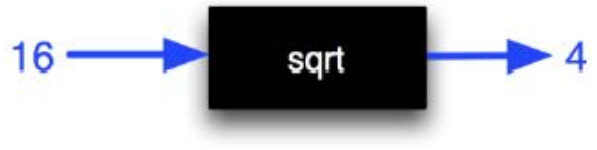
3/ Using “while” statements, write a python program which display prime numbers up to a maximum value max_value

8. Functions

Functions

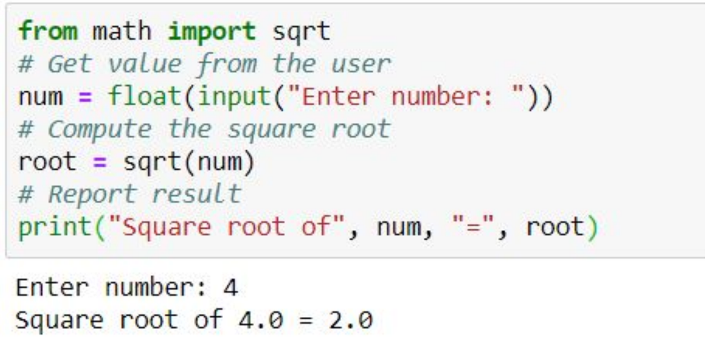
One way to make code more reusable is by packaging it in functions. A function is a unit of reusable code. Python provides a collection of standard functions stored in libraries called modules. Programmers can use the functions from these libraries within their own code to build sophisticated programs.

Conceptual view of the square root function



Using Functions

We have been using functions in Python since the beginning. These functions include print, input, int, float, str, and type. The Python standard library includes many other functions useful for common programming tasks. The example below loads the function sqrt from the math library



```python
from math import sqrt
# Get value from the user
num = float(input("Enter number: "))
# Compute the square root
root = sqrt(num)
# Report result
print("Square root of", num, "=", root)
```

```
Enter number: 4
Square root of 4.0 = 2.0
```

Using Functions

The expression sqrt(num) is a function invocation, also known as a function call.

Unlike the other functions we have used earlier, the interpreter is not automatically aware of the sqrt function. The sqrt function is not part of the small collection of functions (like type, int, and str) always available to Python programs. The sqrt function is part of separate module within the standard library. A module is a collection of Python code that can used in other programs. The import keyword makes a module available to the interpreter

```
from math import sqrt
```

Using Functions

From the caller’s perspective a function has three important parts:

- **Name**. Every function has a name that identifies the code to be executed. Function names follow the same rules as variable names; a function name is another example of an identifier.

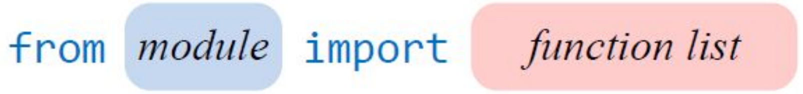
- **Parameters**. A function must be called with a certain number of parameters, and each parameter must be the correct type. If a caller attempts to call a function with too many or too few parameters, the interpreter will issue an error message and refuse to run the program.

- **Result type**. A function returns a value to its caller. Generally a function will compute a result and return the value of the result to the caller. The caller’s use of this result must be compatible with the function’s specified result type. A function’s result type and its parameter types can be completely unrelated. The sqrt function computes and returns a floating-point value.

Functions and modules

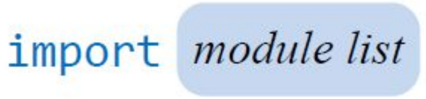
A Python module is simply a file that contains Python code. The name of the file dictates the name of the module; for example, a file named math.py contains the functions available from the standard math module.

- General form of a statement that imports a subset of a module’s available functions



```
from module import function list
```

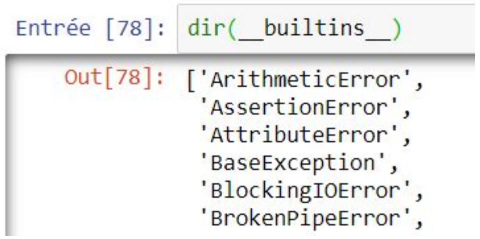
- General form of a statement that imports an entire module



```
import module list
```

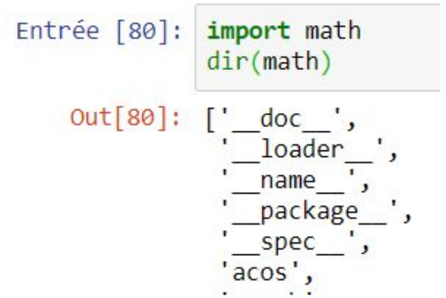
Reveal the list of components of a module

- Built-in functions



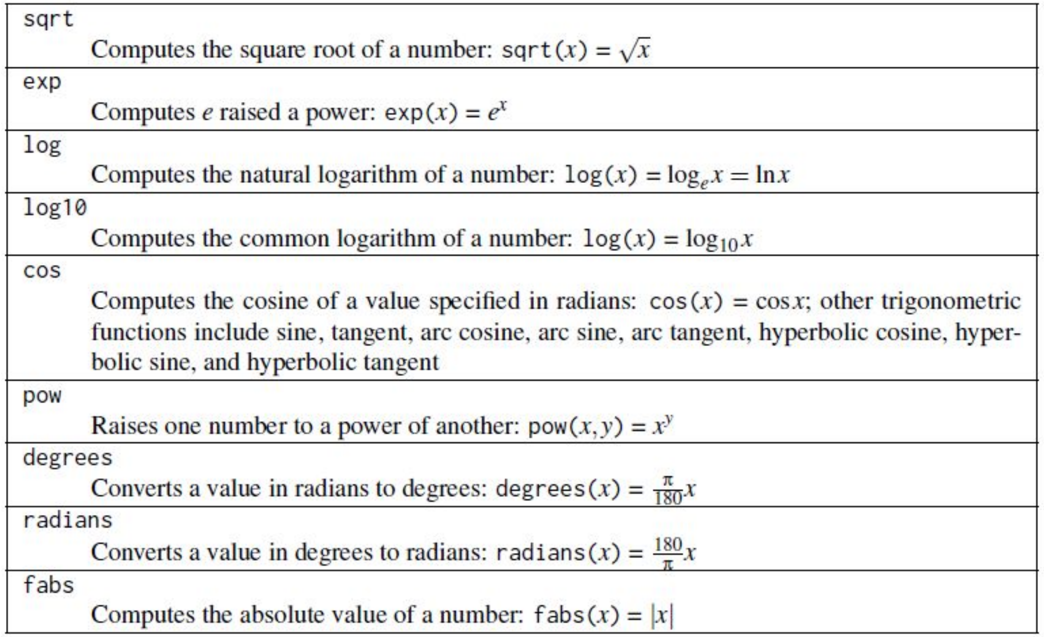
```
Entrée [78]: dir(__builtins__)

   Out[78]: ['ArithmeticError',
             'AssertionError',
             'AttributeError',
             'BaseException',
             'BlockingIOError',
             'BrokenPipeError',
```

- Math module



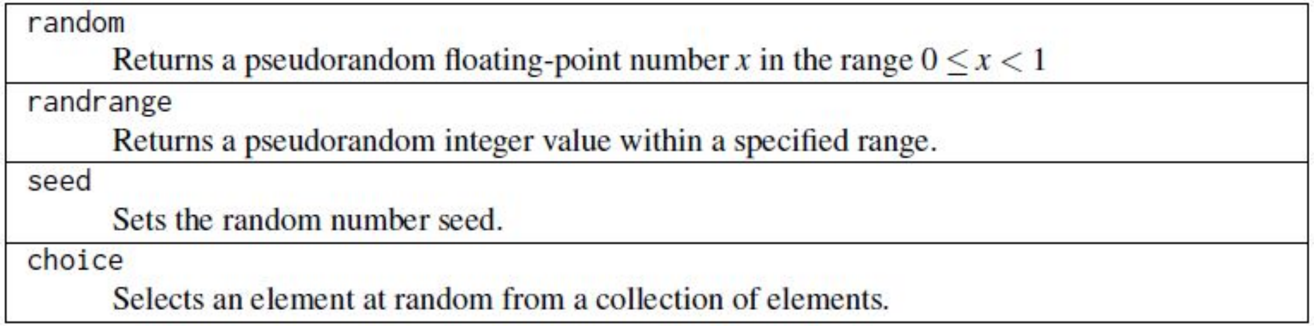
```
Entrée [80]: import math
             dir(math)

   Out[80]: ['__doc__',
             '__loader__',
             '__name__',
             '__package__',
             '__spec__',
             'acos',
```

Standard Mathematical Functions

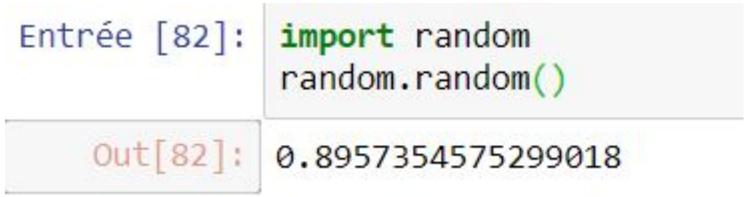


`sqrt`	Computes the square root of a number: `sqrt(`x`)` $= \sqrt{x}$
`exp`	Computes e raised a power: `exp(`x`)` $= e^x$
`log`	Computes the natural logarithm of a number: `log(`x`)` $= \log_e x = \ln x$
`log10`	Computes the common logarithm of a number: `log(`x`)` $= \log_{10} x$
`cos`	Computes the cosine of a value specified in radians: `cos(`x`)` $= \cos x$; other trigonometric functions include sine, tangent, arc cosine, arc sine, arc tangent, hyperbolic cosine, hyperbolic sine, and hyperbolic tangent
`pow`	Raises one number to a power of another: `pow(`x, y`)` $= x^y$
`degrees`	Converts a value in radians to degrees: `degrees(`x`)` $= \frac{\pi}{180} x$
`radians`	Converts a value in degrees to radians: `radians(`x`)` $= \frac{180}{\pi} x$
`fabs`	Computes the absolute value of a number: `fabs(`x`)` $= \lvert x \rvert$

Random module



`random`	Returns a pseudorandom floating-point number x in the range $0 \leq x < 1$
`randrange`	Returns a pseudorandom integer value within a specified range.
`seed`	Sets the random number seed.
`choice`	Selects an element at random from a collection of elements.



```
Entrée [82]: import random
             random.random()

    Out[82]: 0.8957354575299018
```

- The random.seed function establishes the initial value from which the sequence of pseudorandom numbers is generated

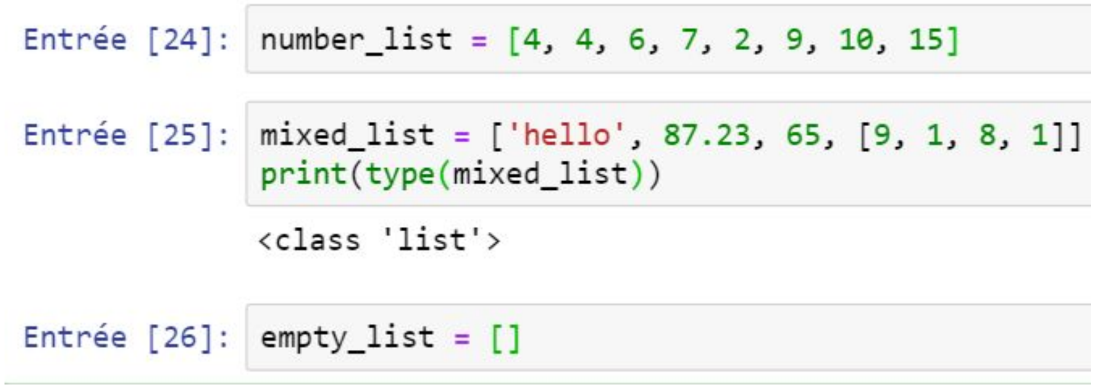
Exercises: Functions

1/ Write a guessing game program in which the computer chooses at random an integer in the range 1: : : 100. The user’s goal is to guess the number in the least number of tries. For each incorrect guess the user provides, the computer provides feedback whether the user’s number is too high or too low.

9. Lists

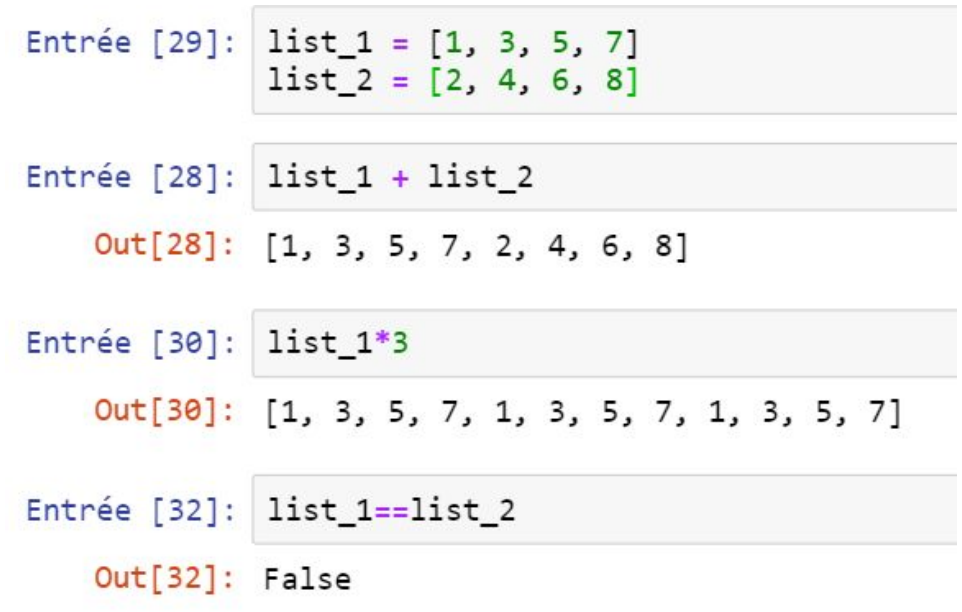
Lists

You can think of the list as a container that holds a number of items. Each element or value that is inside a list is called an item. All the items in a list are assigned to a single variable. Lists avoid having a separate variable to store each item which is less efficient and more error prone when you have to perform some operations on these items



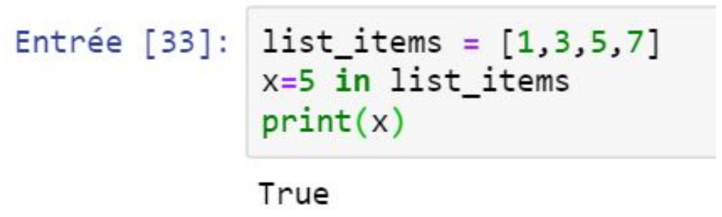
```
Entrée [24]: number_list = [4, 4, 6, 7, 2, 9, 10, 15]

Entrée [25]: mixed_list = ['hello', 87.23, 65, [9, 1, 8, 1]]
             print(type(mixed_list))

             <class 'list'>

Entrée [26]: empty_list = []
```

Basic lists operations



```
Entrée [29]: list_1 = [1, 3, 5, 7]
             list_2 = [2, 4, 6, 8]

Entrée [28]: list_1 + list_2

   Out[28]: [1, 3, 5, 7, 2, 4, 6, 8]

Entrée [30]: list_1*3

   Out[30]: [1, 3, 5, 7, 1, 3, 5, 7, 1, 3, 5, 7]

Entrée [32]: list_1==list_2

   Out[32]: False
```

Basic lists operations



```
Entrée [33]: list_items = [1,3,5,7]
             x=5 in list_items
             print(x)

             True
```

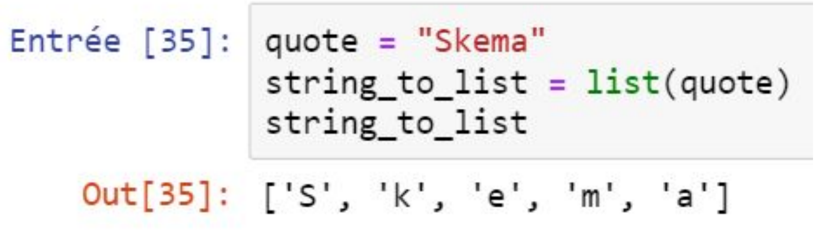
The built-in list() function is used to create a list. The syntax is:

list([sequence])

Where sequence can be a string, a tuple, or a list itself

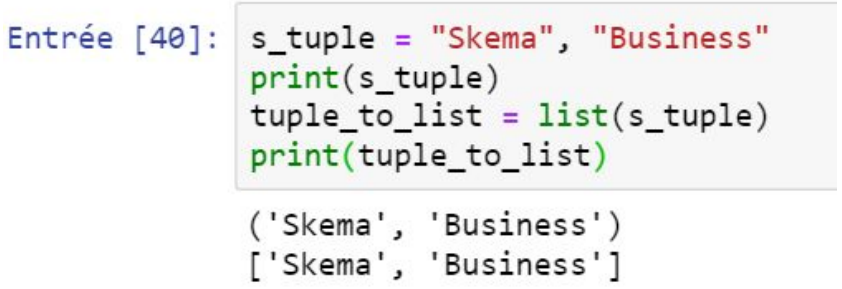
Basic lists operations

- String to list



```
Entrée [35]: quote = "Skema"
             string_to_list = list(quote)
             string_to_list

   Out[35]: ['S', 'k', 'e', 'm', 'a']
```

- Tuple to list

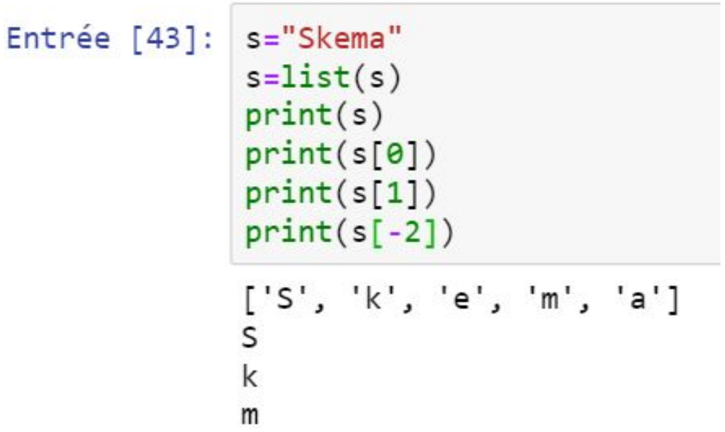


```
Entrée [40]: s_tuple = "Skema", "Business"
             print(s_tuple)
             tuple_to_list = list(s_tuple)
             print(tuple_to_list)

             ('Skema', 'Business')
             ['Skema', 'Business']
```

Indexing and Slicing lists

- Accessing a list: As an ordered sequence of elements, each item in a list can be called individually, through Indexing. The syntax for accessing an item in a list is:

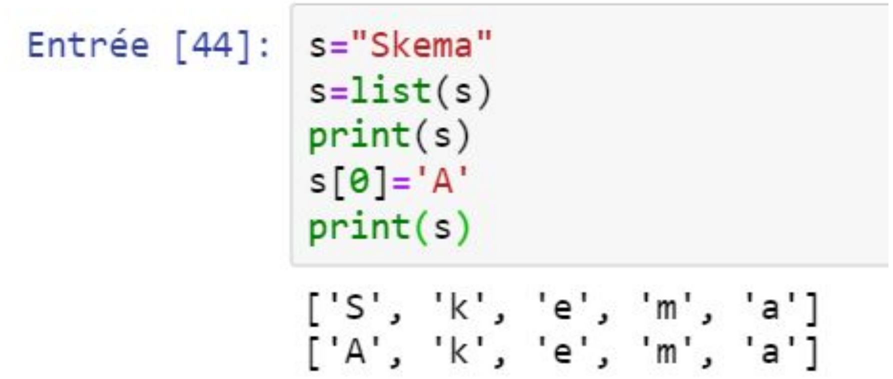
  list_name[index]



```
Entrée [43]: s="Skema"
             s=list(s)
             print(s)
             print(s[0])
             print(s[1])
             print(s[-2])

['S', 'k', 'e', 'm', 'a']
S
k
m
```

Indexing and Slicing lists

- Modifying a list:
  Lists are mutable in nature as the list items can be modified after you have created a list. You can modify a list by replacing the older item with a newer item in its place and without assigning the list to a completely new variable.



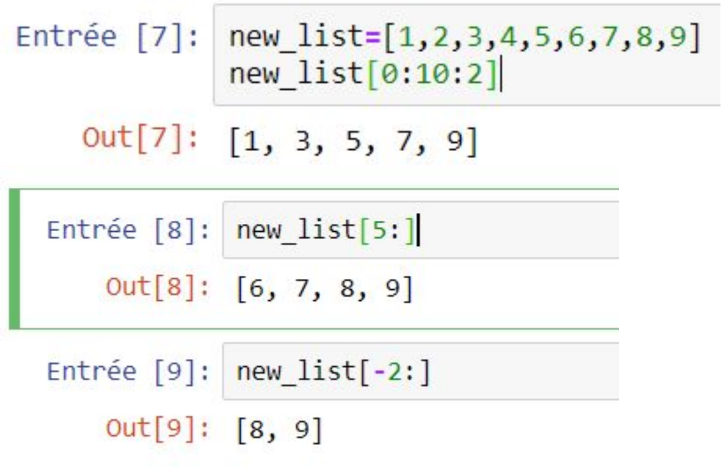
Entrée [44]:
```
s="Skema"
s=list(s)
print(s)
s[0]='A'
print(s)
```

```
['S', 'k', 'e', 'm', 'a']
['A', 'k', 'e', 'm', 'a']
```

Indexing and Slicing lists

- Slicing a list:
  Slicing of lists is allowed in Python. The syntax for list slicing is:

  list_name[start:stop:step]

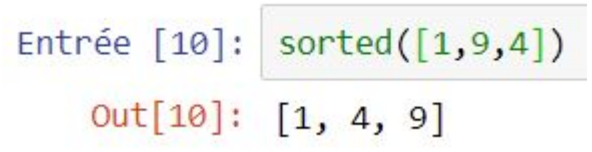


```
Entrée [7]: new_list=[1,2,3,4,5,6,7,8,9]
            new_list[0:10:2]

   Out[7]: [1, 3, 5, 7, 9]

Entrée [8]: new_list[5:]

   Out[8]: [6, 7, 8, 9]

Entrée [9]: new_list[-2:]

   Out[9]: [8, 9]
```

Built-in functions on lists

- Built-In Functions Description

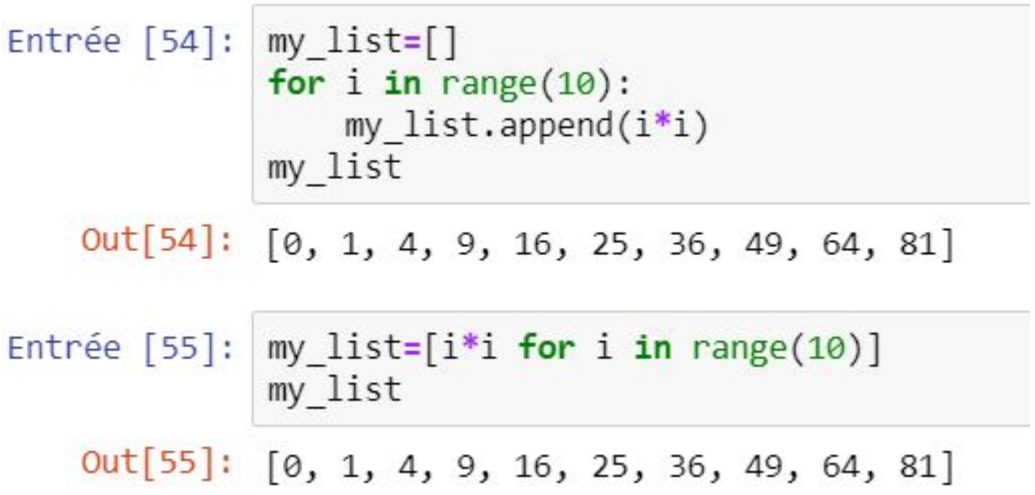
len(): The len() function returns the numbers of items in a list.
sum(): The sum() function returns the sum of numbers in the list.
any(): The any() function returns True if any of the Boolean values in the list is True.
all(): The all() function returns True if all the Boolean values in the list are True, else returns False.
sorted(): The sorted() function returns a modified copy of the list while leaving the original list untouched



```
Entrée [10]: sorted([1,9,4])

   Out[10]: [1, 4, 9]
```

Comprehensive lists

- Also called “computed lists”: list built from an expression on an iterable object.
- Syntax: list_variable = [x for x in iterable]



```
Entrée [54]: my_list=[]
             for i in range(10):
                 my_list.append(i*i)
             my_list

   Out[54]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

Entrée [55]: my_list=[i*i for i in range(10)]
             my_list

   Out[55]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Exercises: lists

1/

- Create a list “greeks” with 3 strings: “delta”, “gamma”, “vega”
- Print the second element of the list
- Change the value from “vega” to “voma” in the list
- Using the append method add the “theta” item to the list
- Use the remove method to remove "delta" from the greeks list

2/

- With the append method, build a list named “list_integers” which contains all integers from 0 to 100
- Repeat the same with a comprehensive list

Exercises: lists

3/

- Create a function unique(L) which removes the duplicate items of a list L

4/

- Write a Python function that takes two lists and returns True if they have at least one common member

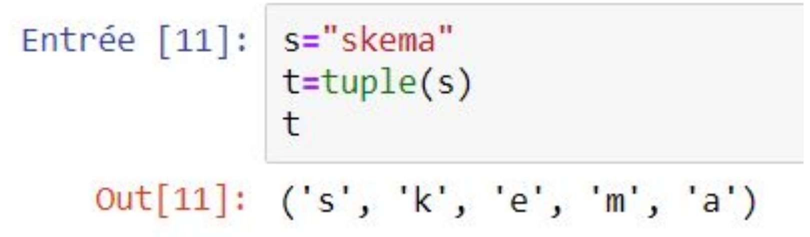
10. Tuples and Sets

Tuples

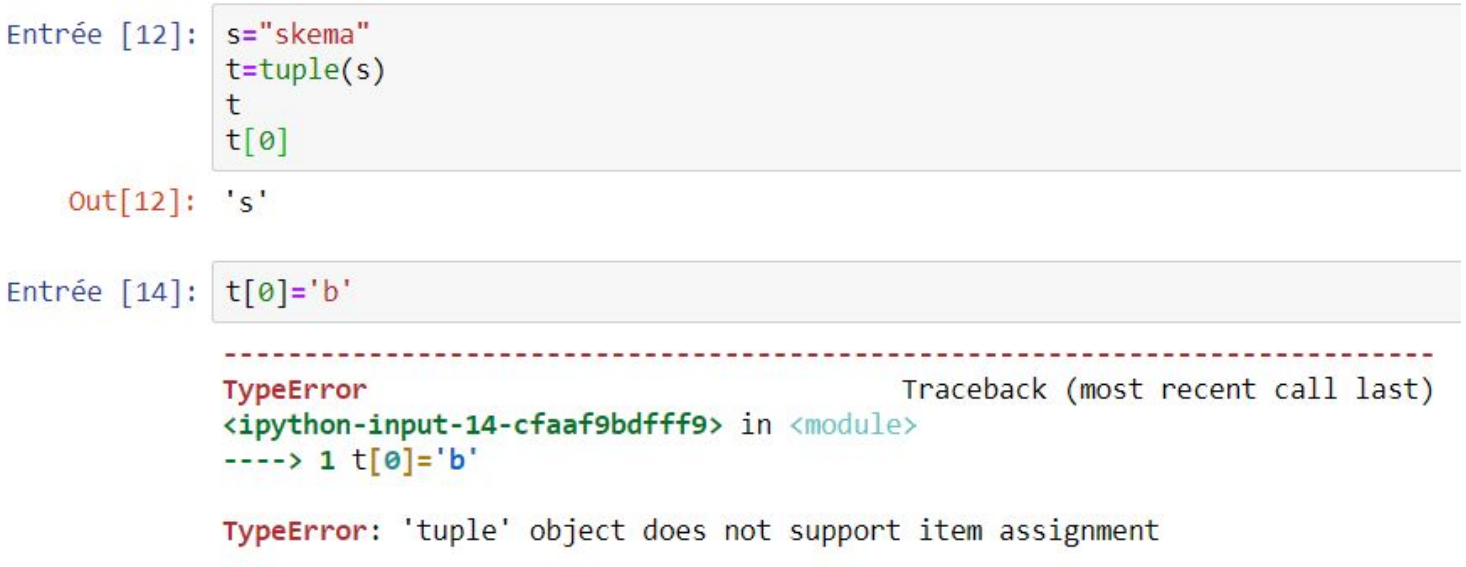
Tuple is a finite ordered list of values of possibly different types which is used to bundle related values together without having to create a specific type to hold them. Tuples are immutable. Once a tuple is created, you cannot change its values. The syntax for creating tuples is:

Tuple = (item1, item2, item3)

Same operations / functions / Slicing methods as lists


```
Entrée [11]: s="skema"
             t=tuple(s)
             t

    Out[11]: ('s', 'k', 'e', 'm', 'a')
```

Tuples are immutable

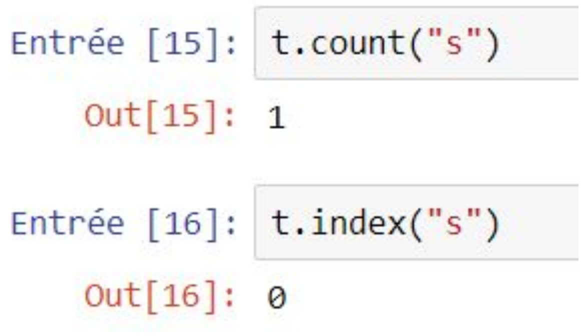


```
Entrée [12]: s="skema"
             t=tuple(s)
             t
             t[0]

   Out[12]: 's'

Entrée [14]: t[0]='b'

             ---------------------------------------------------------------------------
             TypeError                                 Traceback (most recent call last)
             <ipython-input-14-cfaaf9bdfff9> in <module>
             ----> 1 t[0]='b'

             TypeError: 'tuple' object does not support item assignment
```

Useful tuple methods

count() tuple_name.count(item): The count() method counts the number of times the item has occurred in the tuple and returns it.

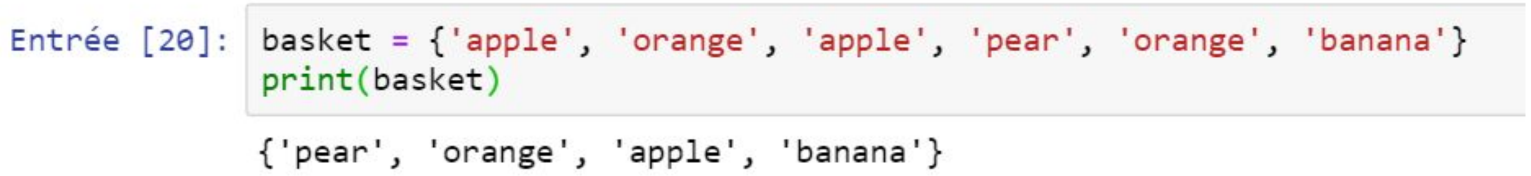
index() tuple_name.index(item) The index() method searches for the given item from the start of the tuple and returns its index. If the value appears more than once, you will get the index of the first one. If the item is not present in the tuple, then ValueError is thrown by this method



```
Entrée [15]: t.count("s")

   Out[15]: 1

Entrée [16]: t.index("s")

   Out[16]: 0
```

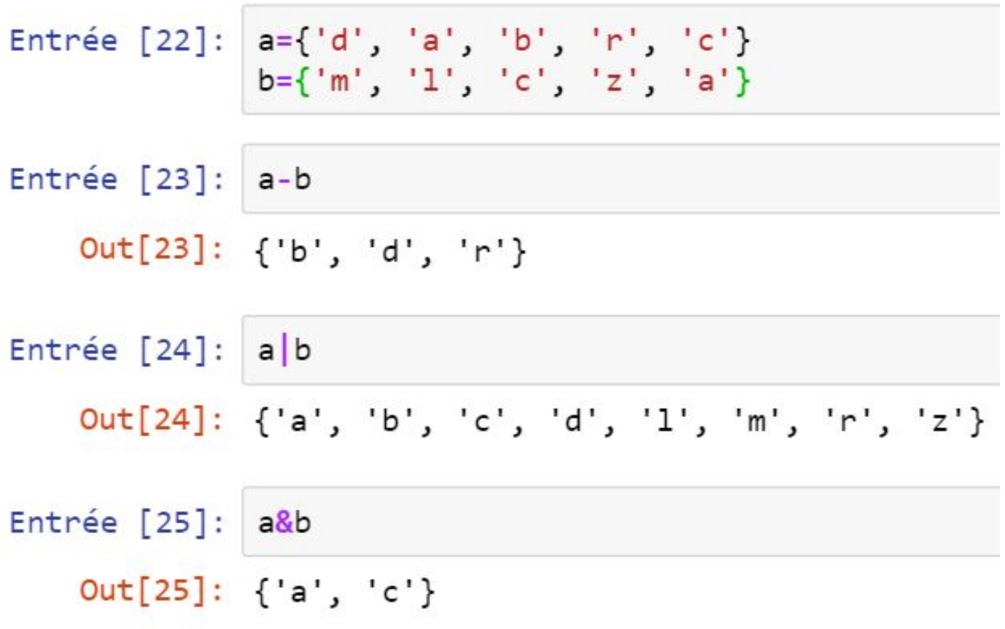
Sets

Python also includes a data type for sets. A set is an unordered collection with no duplicate
Items. Primary uses of sets include membership testing and eliminating duplicate
entries



Entrée [20]:
```
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
print(basket)
```
```
{'pear', 'orange', 'apple', 'banana'}
```

Sets for intersection/union

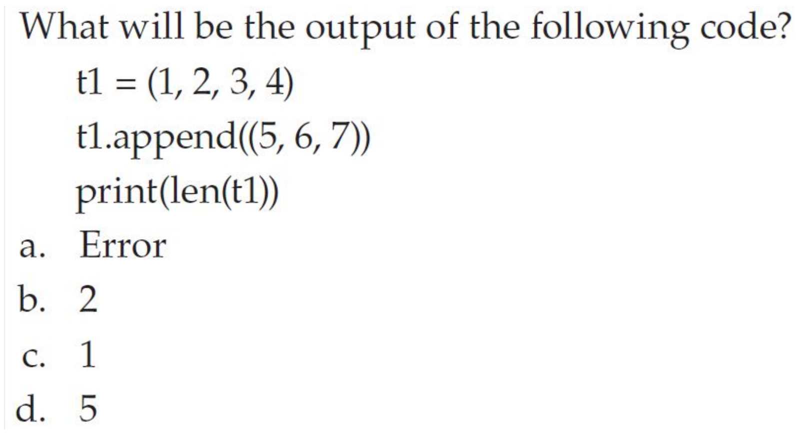


```
Entrée [22]: a={'d', 'a', 'b', 'r', 'c'}
             b={'m', 'l', 'c', 'z', 'a'}

Entrée [23]: a-b

    Out[23]: {'b', 'd', 'r'}

Entrée [24]: a|b

    Out[24]: {'a', 'b', 'c', 'd', 'l', 'm', 'r', 'z'}

Entrée [25]: a&b

    Out[25]: {'a', 'c'}
```

Exercises: Tuples and sets

1/ Write Python Program to Sort Words in a Sentence in Decreasing Order of Their Length. Display the Sorted Words along with Their Length

2/ Using sets, create a function Unique which removes the duplicates of a list L

3/ 

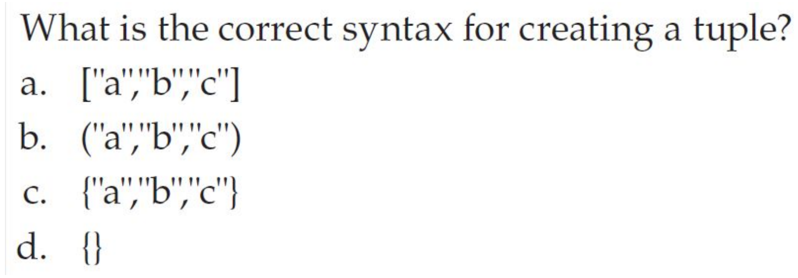
What will be the output of the following code?

```
t1 = (1, 2, 3, 4)
t1.append((5, 6, 7))
print(len(t1))
```

a. Error
b. 2
c. 1
d. 5

Exercises: Tuples and sets

4/



What is the correct syntax for creating a tuple?

a. ["a","b","c"]

b. ("a","b","c")

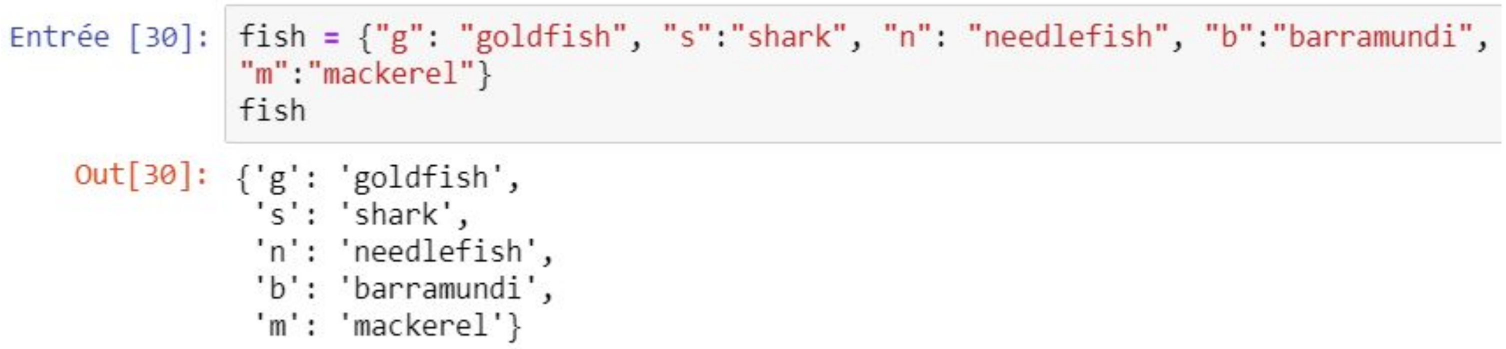
c. {"a","b","c"}

d. {}

11. Dictionaries

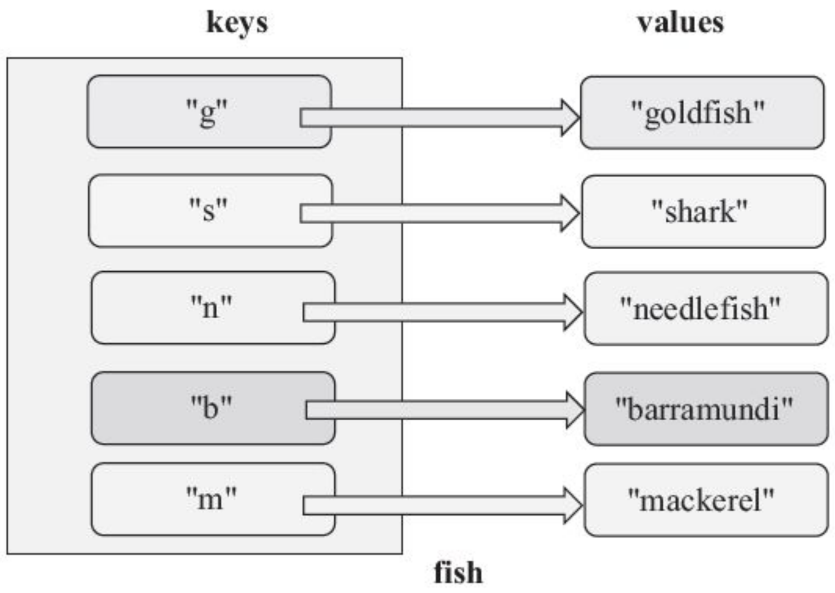
Dictionaries

Another useful data type built into Python is the Dictionary. A dictionary is a collection of an unordered set of key:value pairs, with the requirement that the keys are unique within a dictionary. Dictionaries are constructed using curly braces { }, wherein you include a list of key:value pairs separated by commas



```
Entrée [30]: fish = {"g": "goldfish", "s":"shark", "n": "needlefish", "b":"barramundi",
             "m":"mackerel"}
             fish

Out[30]: {'g': 'goldfish',
          's': 'shark',
          'n': 'needlefish',
          'b': 'barramundi',
          'm': 'mackerel'}
```

Dictionaries



Accessing and Modifying key:value Pairs in Dictionaries

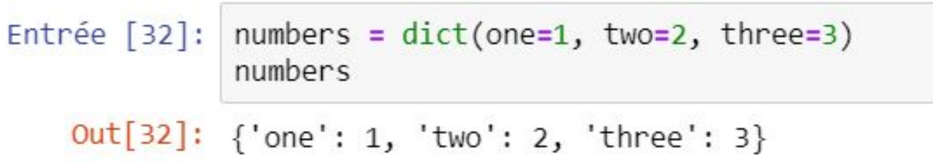
The syntax for accessing the value for a key in the dictionary is:

Dictionary_name[key]

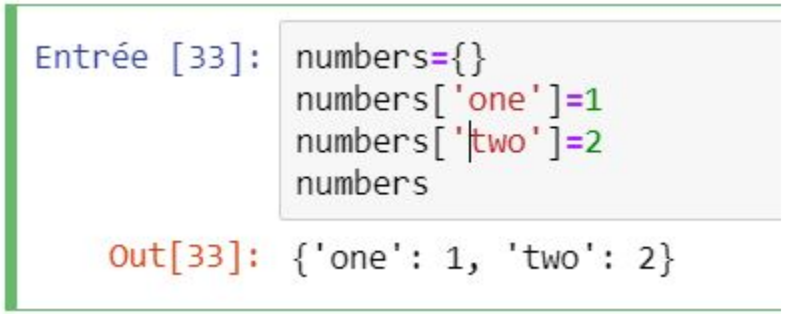
The syntax for modifying the value of an existing key or for adding a new key:value pair to a dictionary is

dictionary_name[key] = value

Creating a dictionary



```
Entrée [32]: numbers = dict(one=1, two=2, three=3)
             numbers

    Out[32]: {'one': 1, 'two': 2, 'three': 3}
```

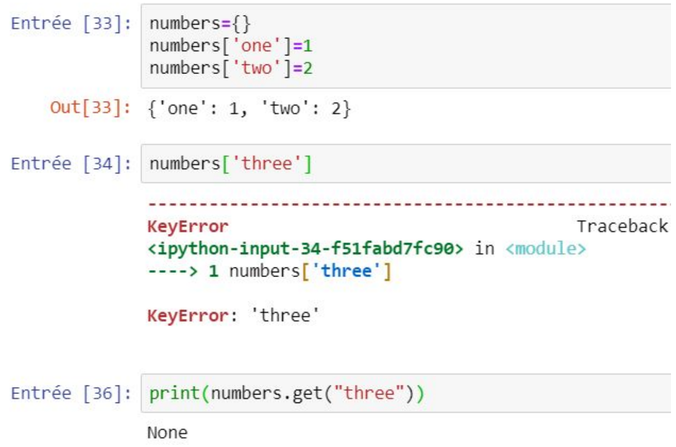


```
Entrée [33]: numbers={}
             numbers['one']=1
             numbers['two']=2
             numbers

    Out[33]: {'one': 1, 'two': 2}
```

Interesting dictionary methods

clear(): The clear() method removes all the key:value pairs from the dictionary

get(): The get() method returns the value associated with the specified key in the dictionary. If the key is not present then it returns the default value. If default is not given, it defaults to None, so that this method never raises a KeyError



```
Entrée [33]: numbers={}
             numbers['one']=1
             numbers['two']=2

    Out[33]: {'one': 1, 'two': 2}

Entrée [34]: numbers['three']

             ---------------------------------------------------------
             KeyError                                        Traceback
             <ipython-input-34-f51fabd7fc90> in <module>
             ----> 1 numbers['three']

             KeyError: 'three'

Entrée [36]: print(numbers.get("three"))

             None
```

Interesting dictionary methods

items(): The items() method returns a new view of dictionary’s key and value pairs as tuples

keys(): The keys() method returns a new view consisting of all the keys in the Dictionary

values(): The values() method returns a new view consisting of all the values in the dictionary

Mutable vs Immutable

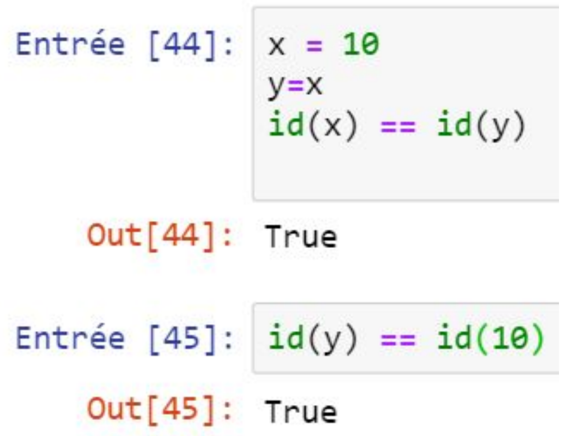
Simple put, a mutable object can be changed after it has been created, and an immutable object can’t.

Mutable objects:
list, dict, set

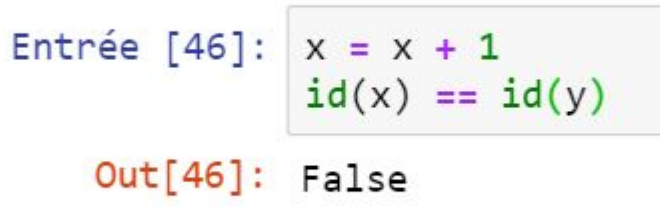
Immutable objects:
int, float, string, tuple

Mutable vs Immutable

Example of immutable



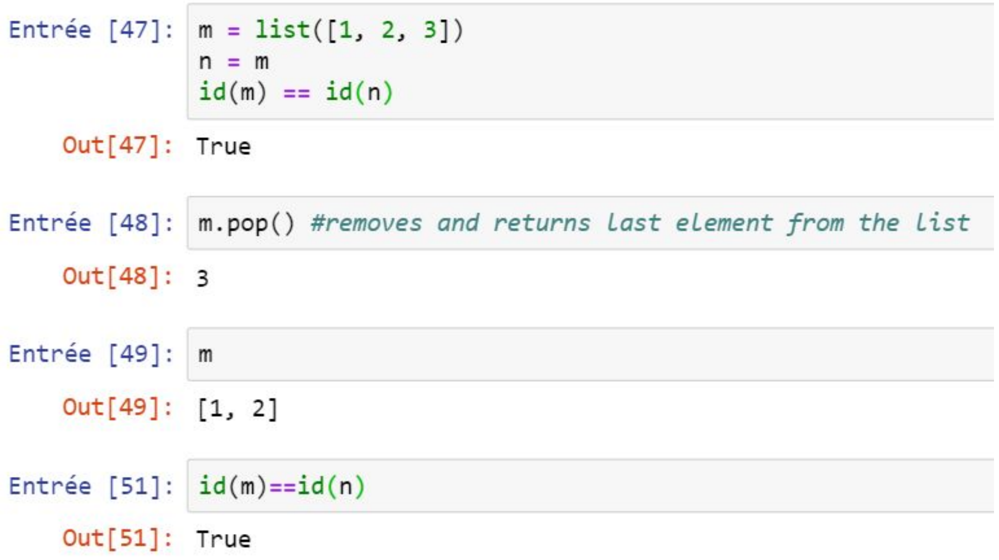
```
Entrée [44]: x = 10
             y=x
             id(x) == id(y)

   Out[44]: True

Entrée [45]: id(y) == id(10)

   Out[45]: True
```



```
Entrée [46]: x = x + 1
             id(x) == id(y)

   Out[46]: False
```

Mutable vs Immutable

Example mutable



```
Entrée [47]: m = list([1, 2, 3])
             n = m
             id(m) == id(n)

    Out[47]: True

Entrée [48]: m.pop() #removes and returns last element from the list

    Out[48]: 3

Entrée [49]: m

    Out[49]: [1, 2]

Entrée [51]: id(m)==id(n)

    Out[51]: True
```

Exercises: Dictionaries

1/ Build a dictionary named “equity” with
- keys: “stock.price”, “stock.volatility”, “stock.code”
- Values: 2500, 0.1, “GOOG”

WIth the get method print the content of the “stock.price” key
Change stock price from 2500 to 3500
Add the key/value “stock.list_indices”/[”DowJones”, “Nasdaq”] to the dictionary
Print the dictionary keys and values.

2/ Write Python Program to Check for the Presence of a Key in the Dictionary and to Sum All Its Values

Exercises: Dictionaries

3/ Which of the following cannot be used as a key in Python dictionaries?

a. Strings
b. Lists
c. Tuples
d. Numerical values

4/ Write a Python program to combine two dictionary adding values for common keys.

```
d1 = {'a': 100, 'b': 200, 'c':300}
d2 = {'a': 300, 'b': 200, 'd':400}
```

12. File processing

File processing

Many programs handle data, which often comes from files. Reading the entire contents of a file:

variableName = open("filename").read()
Example:
```
file_text = open("bankaccount.txt").read()
```

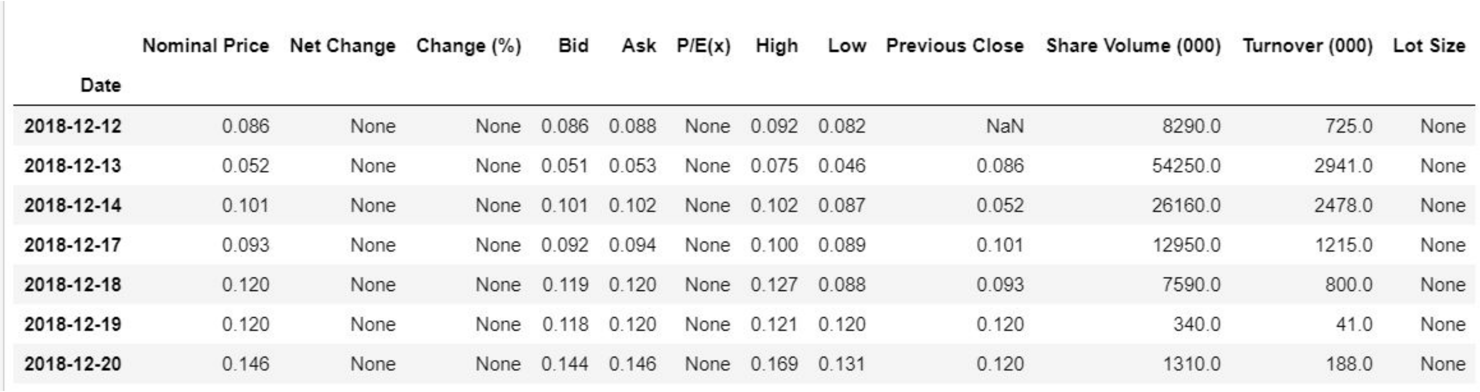
Reading a file line-by-line:

for line in open("***filename***").readlines():
    statements

Example:
```
count = 0
for line in open("bankaccount.txt").readlines():
    count = count + 1
print "The file contains", count, "lines."
```

Quandl

```
import sys
!{sys.executable} -m pip install quandl

import quandl
x=quandl.get("HKEX/58536", authtoken="zbzphcQYstaEEmzXfs_F")
```



```
            Nominal Price  Net Change  Change (%)    Bid    Ask  P/E(x)   High    Low  Previous Close  Share Volume (000)  Turnover (000)  Lot Size
      Date
2018-12-12          0.086        None        None  0.086  0.088    None  0.092  0.082             NaN              8290.0           725.0      None
2018-12-13          0.052        None        None  0.051  0.053    None  0.075  0.046           0.086             54250.0          2941.0      None
2018-12-14          0.101        None        None  0.101  0.102    None  0.102  0.087           0.052             26160.0          2478.0      None
2018-12-17          0.093        None        None  0.092  0.094    None  0.100  0.089           0.101             12950.0          1215.0      None
2018-12-18          0.120        None        None  0.119  0.120    None  0.127  0.088           0.093              7590.0           800.0      None
2018-12-19          0.120        None        None  0.118  0.120    None  0.121  0.120           0.120               340.0            41.0      None
2018-12-20          0.146        None        None  0.144  0.146    None  0.169  0.131           0.120              1310.0           188.0      None
```