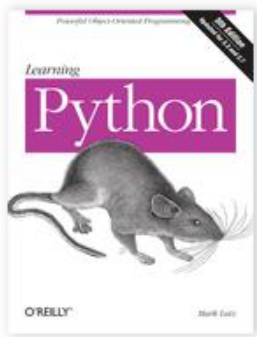


# Python

# Referencias



## Learning Python, 5th Edition

by **Mark Lutz**

Publisher: **O'Reilly Media, Inc.**

Release Date: *June 2013*

ISBN: 9781449355739

Topic: **Python**



## Jupyter for Data Science

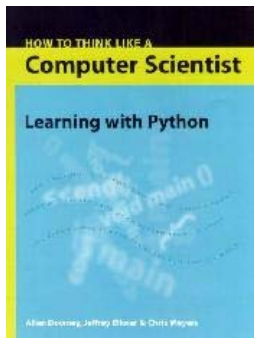
by **Dan Toomey**

Publisher: **Packt Publishing**

Release Date: *October 2017*

ISBN: 9781785880070

Topic: **Jupyter**



## How to think like a Computer Scientist Learning with Python

by **Allen Downey, Jeff Elkner and Chris Meyers.**

GNU free document

<https://greenteapress.com/wp/learning-with-python/>



## Python Cookbook, 3rd Edition

by **David Beazley, Brian K. Jones**

Publisher: **O'Reilly Media, Inc.**

Release Date: *May 2013*

ISBN: 9781449340377

Topic: **Python**



# Anaconda Development Platform

Variables, Expressions & Operators

Data Types

Conditionals

Loops

Functions

Files



Windows



macOS



Linux

## Anaconda 2019.07 for Windows Installer

### Python 3.7 version

Download

64-Bit Graphical Installer (486 MB)

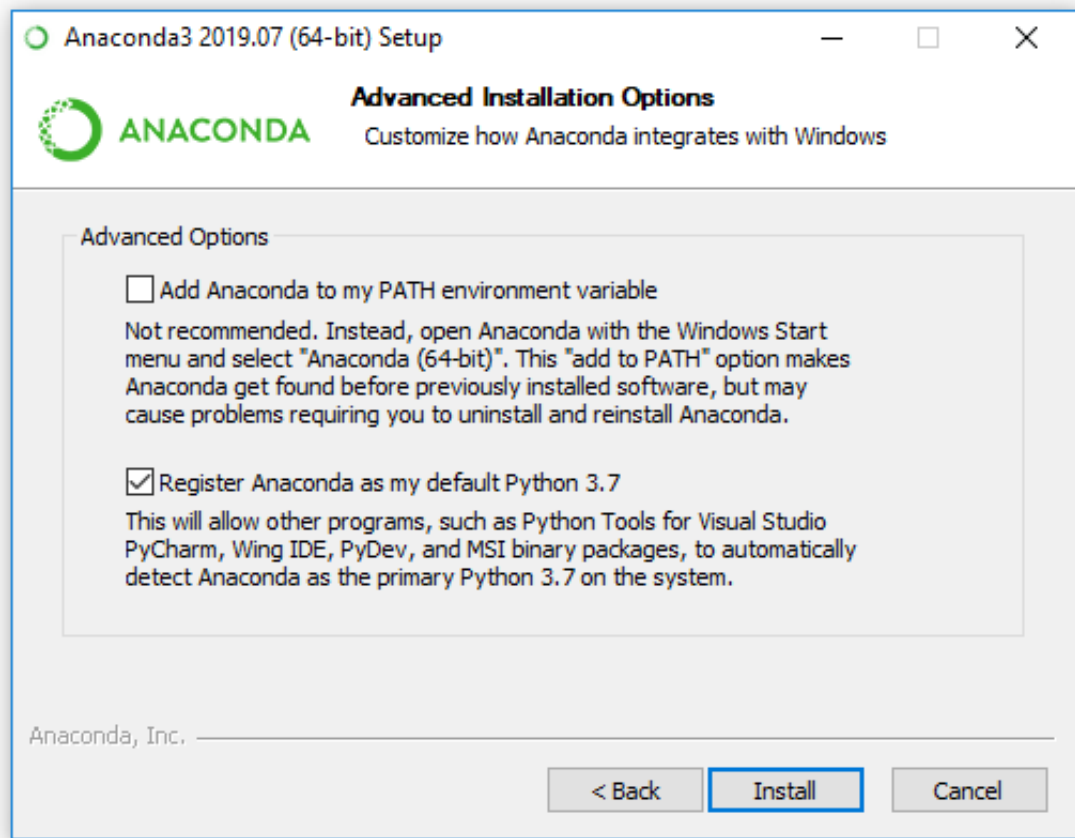
32-Bit Graphical Installer (418 MB)

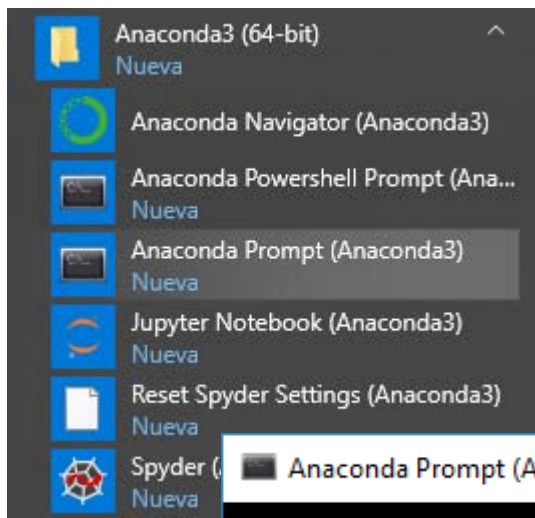
### Python 2.7 version

Download

64-Bit Graphical Installer (427 MB)

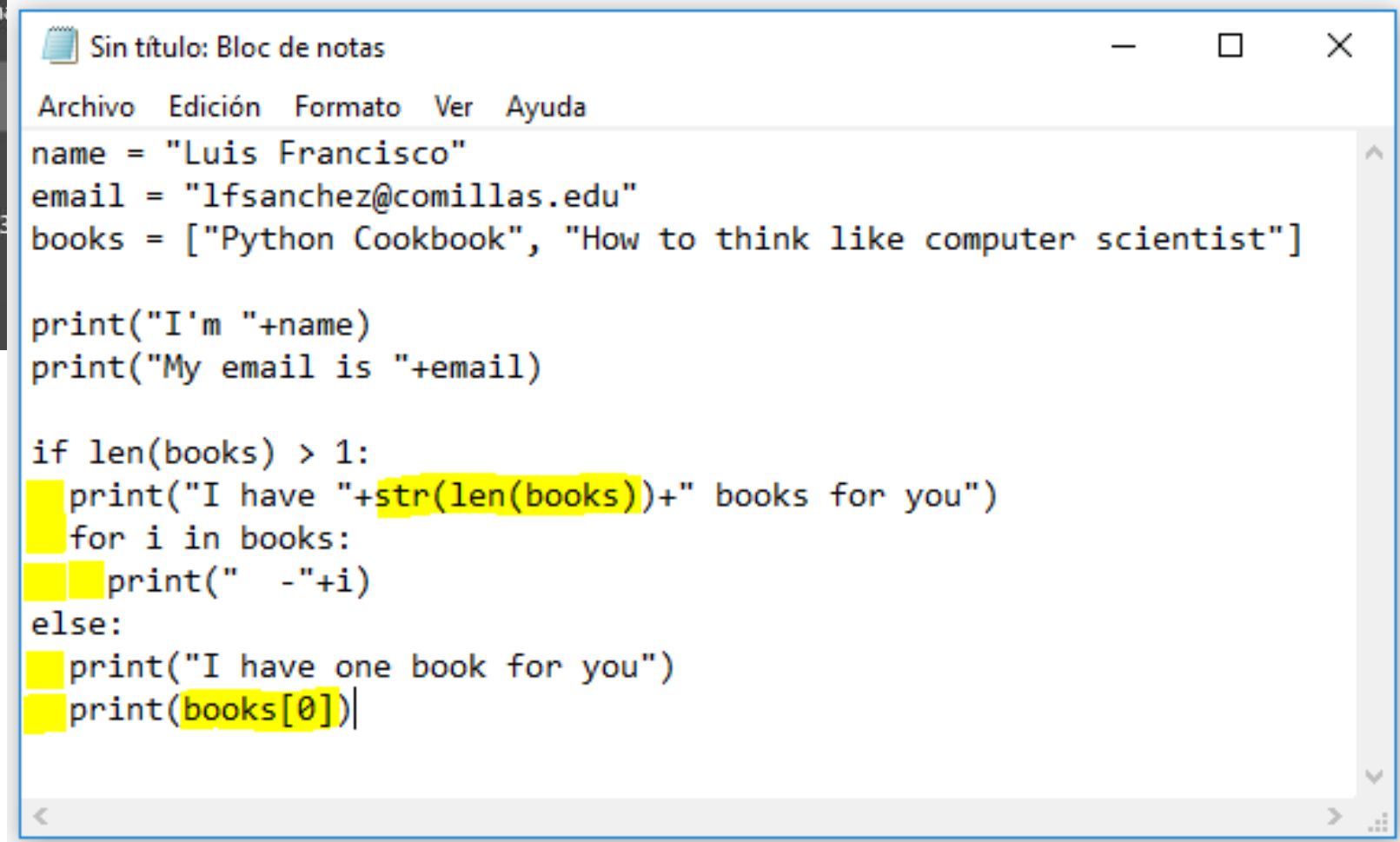
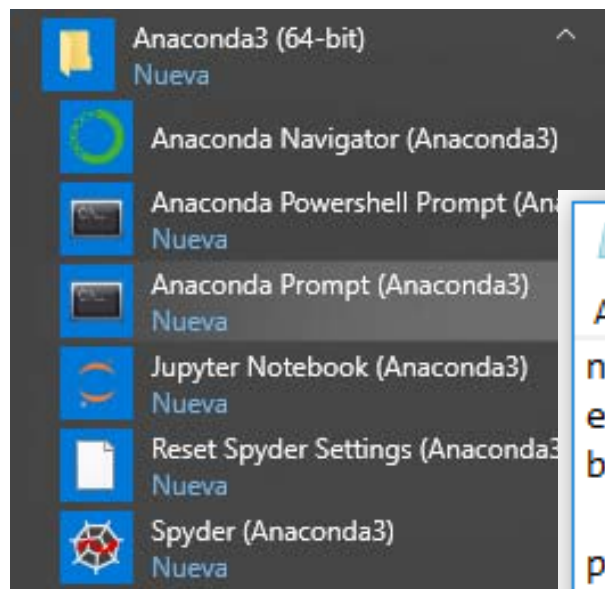
32-Bit Graphical Installer (361 MB)





We are invoke the Python  
command line

```
Anaconda Prompt (Anaconda3) - python
(base) C:\Users\luis>python
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> name = "Luis Francisco"
>>> email = "lfsanchez@comillas.edu"
>>> books = ["Python Cookbook", "How to think like computer scientist"]
>>> print("I'm "+name+".\n"+"My email is "+email+".\n"+"Two valid references are: "+" and '.join(books))
I'm Luis Francisco.
My email is lfsanchez@comillas.edu.
Two valid references are: Python Cookbook and How to think like computer scientist
>>> _
```

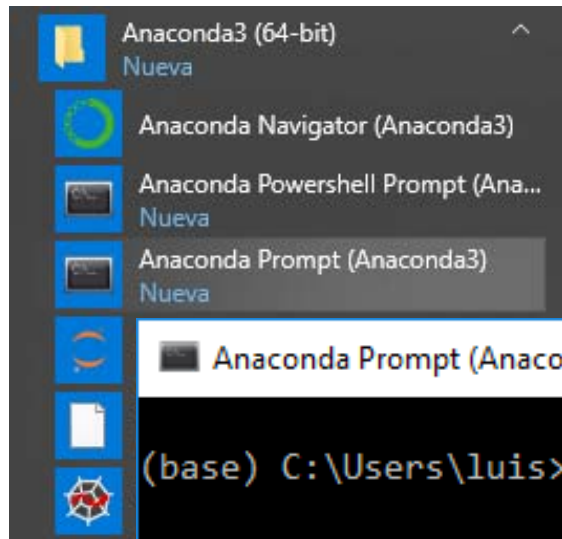
A screenshot of a Windows Notepad window titled 'Sin título: Bloc de notas'. The window has a standard menu bar with 'Archivo', 'Edición', 'Formato', 'Ver', and 'Ayuda'. The text area contains Python code. The code defines variables for a name, email, and a list of books. It then prints the name and email. A conditional statement checks the length of the books list. If there are more than one book, it prints the count and lists each book. Otherwise, it prints a message about having one book and prints the first book. The code is as follows:

```
Sin título: Bloc de notas
Archivo  Edición  Formato  Ver  Ayuda

name = "Luis Francisco"
email = "lfsanchez@comillas.edu"
books = ["Python Cookbook", "How to think like computer scientist"]

print("I'm "+name)
print("My email is "+email)

if len(books) > 1:
    print("I have "+str(len(books))+" books for you")
    for i in books:
        print("  -"+i)
else:
    print("I have one book for you")
    print(books[0])
```

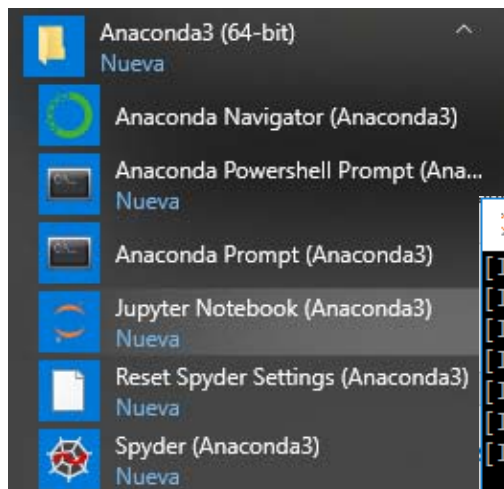


We don't enter into the Python command line

A screenshot of an Anaconda Prompt window. The window title is 'Anaconda Prompt (Anaconda3)'. The command prompt shows the following sequence of commands and output:

```
(base) C:\Users\luis>cd "C:\Users\luis\OneDrive\ICADE\LuisFco\Python\Clases\Resources Chapter 1\"
(base) C:\Users\luis\OneDrive\ICADE\LuisFco\Python\Clases\Resources Chapter 1>python start.py
I'm Luis Francisco
My email is lfsanchez@comillas.edu
I have 2 books for you
  -Python Cookbook
  -How to think like computer scientist
(base) C:\Users\luis\OneDrive\ICADE\LuisFco\Python\Clases\Resources Chapter 1>
```

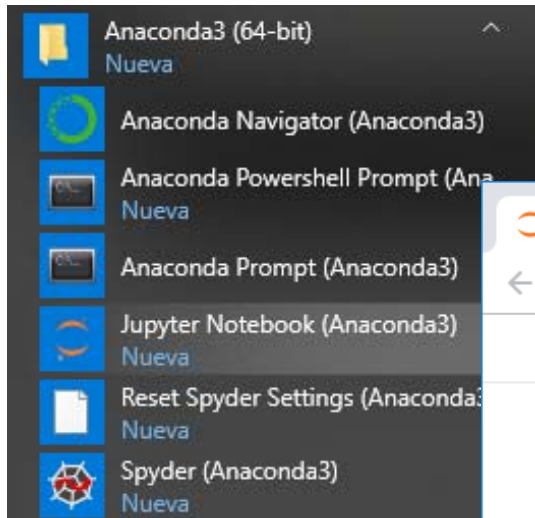




```
Jupyter Notebook (Anaconda3)
[I 21:30:03.962 NotebookApp] JupyterLab extension loaded from C:\Users\luis\Anaconda3\lib\site-packages\jupyterlab
[I 21:30:03.962 NotebookApp] JupyterLab application directory is C:\Users\luis\Anaconda3\share\jupyter\lab
[I 21:30:03.965 NotebookApp] Serving notebooks from local directory: C:\Users\luis
[I 21:30:03.965 NotebookApp] The Jupyter Notebook is running at:
[I 21:30:03.966 NotebookApp] http://localhost:8888/?token=81d6b0c9365d3ffb538e4f250a7ebaafa9ee70c8a25f5bc8
[I 21:30:03.966 NotebookApp] or http://127.0.0.1:8888/?token=81d6b0c9365d3ffb538e4f250a7ebaafa9ee70c8a25f5bc8
[I 21:30:03.966 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).

[C 21:30:04.066 NotebookApp]

To access the notebook, open this file in a browser:
    file:///C:/Users/luis/AppData/Roaming/jupyter/runtime/nbserver-10036-open.html
Or copy and paste one of these URLs:
    http://localhost:8888/?token=81d6b0c9365d3ffb538e4f250a7ebaafa9ee70c8a25f5bc8
    or http://127.0.0.1:8888/?token=81d6b0c9365d3ffb538e4f250a7ebaafa9ee70c8a25f5bc8
[E 21:30:05.512 NotebookApp] Could not open static file ''
[W 21:30:05.612 NotebookApp] 404 GET /static/components/react/react-dom.production.min.js (::1) 17.36ms referer=http://localhost:8888/tree?token=81d6b0c9365d3ffb538e4f250a7ebaafa9ee70c8a25f5bc8
[W 21:30:05.653 NotebookApp] 404 GET /static/components/react/react-dom.production.min.js (::1) 4.46ms referer=http://localhost:8888/tree?token=81d6b0c9365d3ffb538e4f250a7ebaafa9ee70c8a25f5bc8
```



Home Page - Select or create a notebook

localhost:8888/tree

jupyter

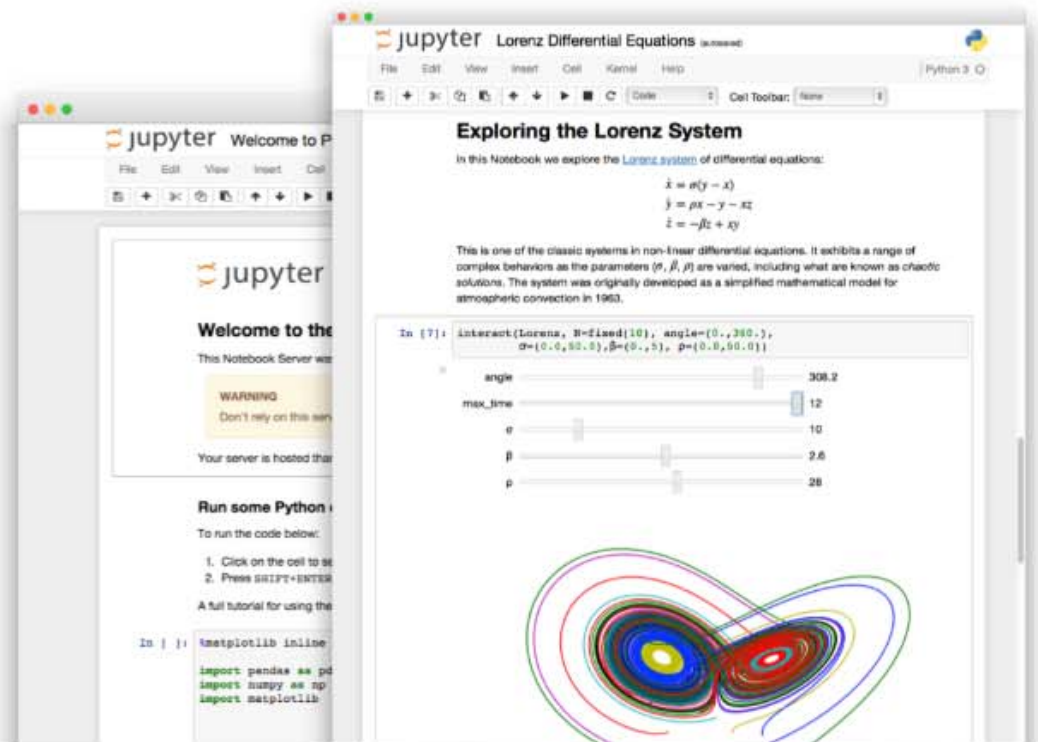
Quit Logout

Files Running Clusters

Select items to perform actions on them.

Upload New ↕

<input type="checkbox"/> 0	Name ↓	Last Modified	File size
<input type="checkbox"/>	3D Objects	hace un mes	
<input type="checkbox"/>	Anaconda2	hace 3 meses	
<input type="checkbox"/>	Anaconda3	hace 40 minutos	
<input type="checkbox"/>	Application Data	hace 8 meses	
<input type="checkbox"/>	Contacts	hace un mes	
<input type="checkbox"/>	Desktop	hace 38 minutos	
<input type="checkbox"/>	Documents	hace un mes	
<input type="checkbox"/>	Downloads	hace una hora	
<input type="checkbox"/>	Dropbox	hace un mes	
<input type="checkbox"/>	Favorites	hace un mes	
<input type="checkbox"/>	Google Drive	hace 2 horas	



## The Jupyter Notebook


The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

[Try it in your browser](#)[Install the Notebook](#)

Home Page - Select or create a notebook

localhost:8888/tree

★🔒📄⬇️📁👤⋮



QuitLogout

FilesRunningClusters

Select items to perform actions on them.

☐ 0 ▾ 📁 /

Name ▾

UploadNew↻

☐ 📁 3D Objects

☐ 📁 Anaconda2

☐ 📁 Anaconda3

☐ 📁 Application Data

☐ 📁 ...

Notebook:  
Python 3

Create a new notebook with Python 3  
Text File  
Folder  
Terminal

hace un mes


localhost:8888/tree#


Untitled - Jupyter Notebook

+

localhost:8888/notebooks/OneDrive/ICADE/LuisFco/Python/C...

☆

 **jupyter** **Untitled** (unsaved changes)



Logout

File

Edit

View

Insert

Cell













Kernel

Widgets


Help

Trusted

Python 3

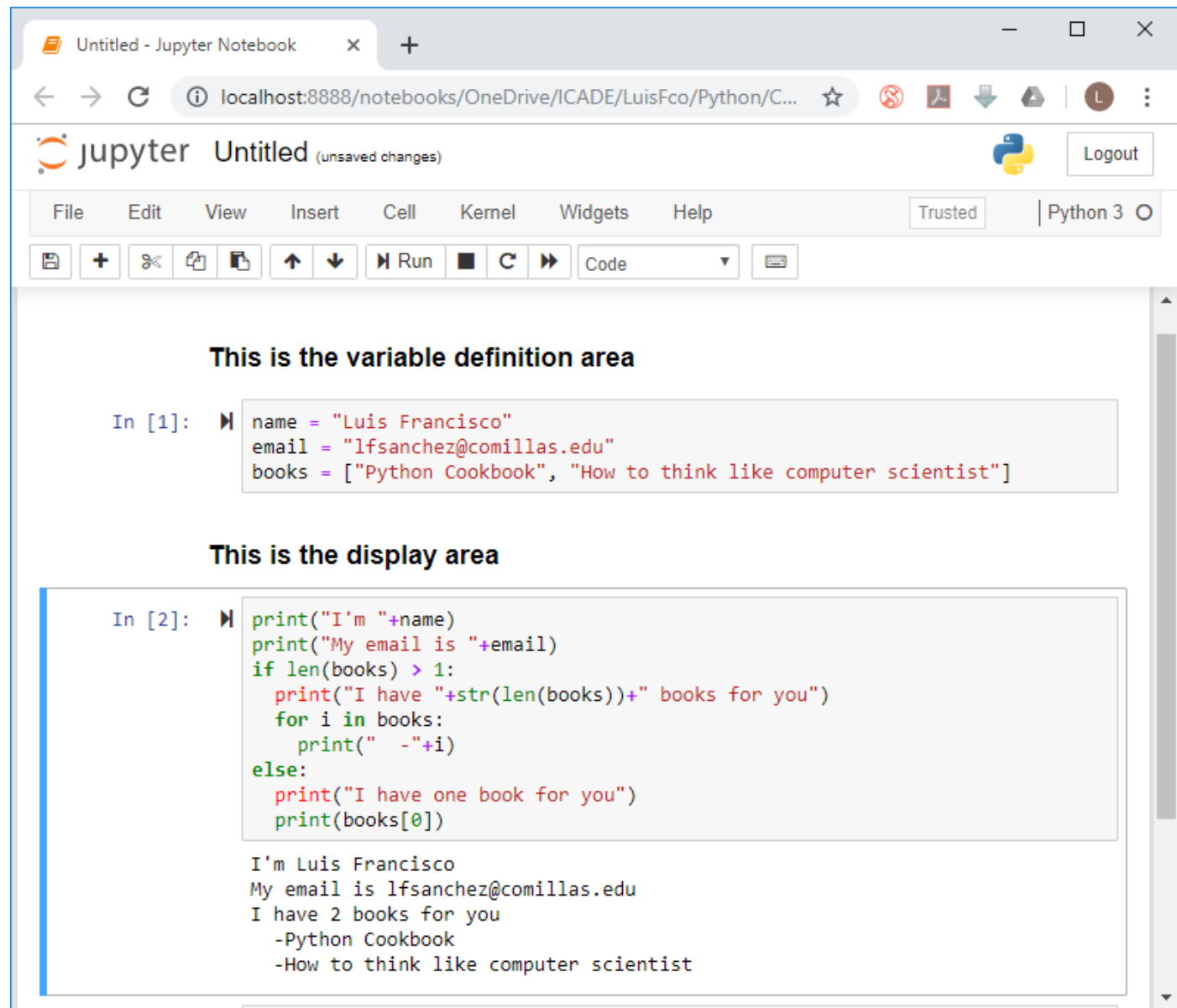
        Run    Code 

In [ ]:



# Jupyter Notebooks

- Split in cells
- Modify previous values
- Introduce comments



Untitled - Jupyter Notebook

localhost:8888/notebooks/OneDrive/ICADE/LuisFco/Python/C...

jupyter Untitled (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run

**This is the variable definition area**

```
In [1]: name = "Luis Francisco"
        email = "lfsanchez@comillas.edu"
        books = ["Python Cookbook", "How to think like computer scientist"]
```

**This is the display area**

```
In [2]: print("I'm "+name)
        print("My email is "+email)
        if len(books) > 1:
            print("I have "+str(len(books))+ " books for you")
            for i in books:
                print(" -"+i)
        else:
            print("I have one book for you")
            print(books[0])
```

I'm Luis Francisco  
My email is lfsanchez@comillas.edu  
I have 2 books for you  
-Python Cookbook  
-How to think like computer scientist

# Jupyter Kernels

- There exists kernels for many languages

- Installing kernels:

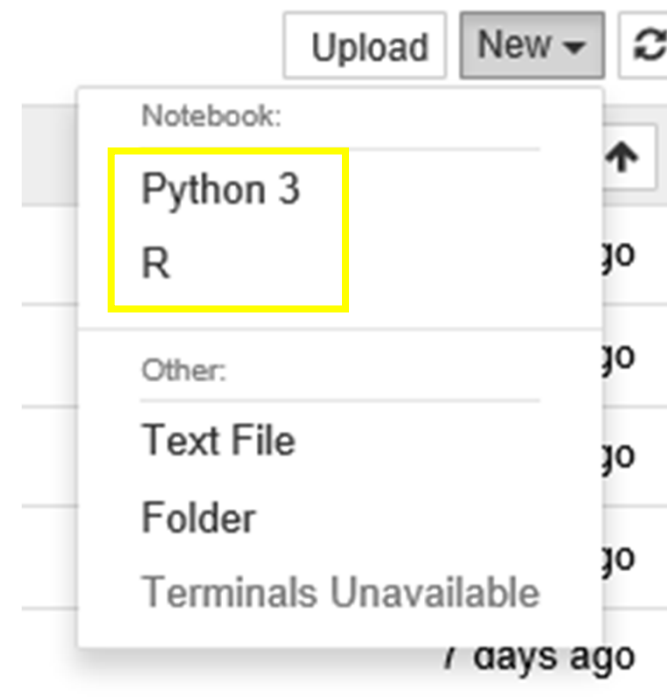
<https://jupyter.readthedocs.io/en/latest/install-kernel.html>

- Available kernels:

<https://github.com/jupyter/jupyter/wiki/Jupyter-kernels>

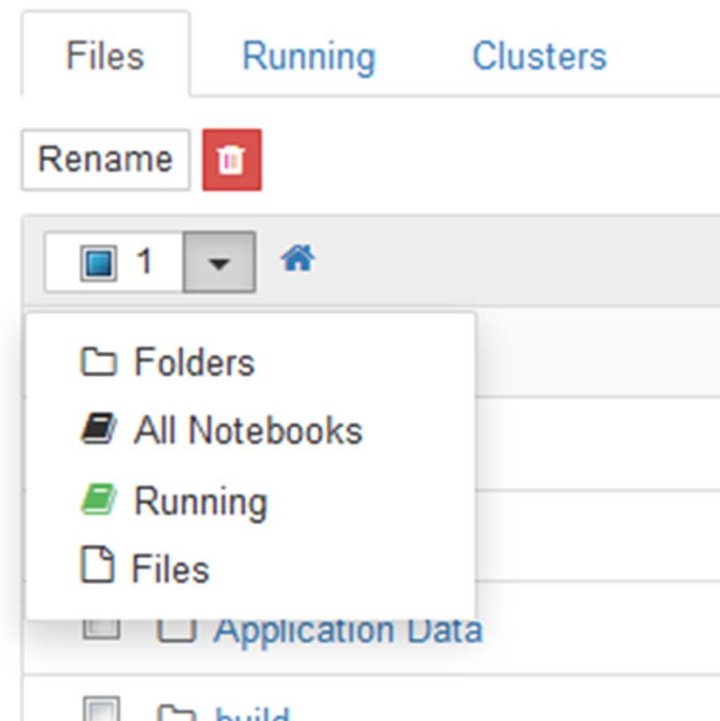
- Create your own kernels:

<https://jupyter-client.readthedocs.io/en/latest/kernels.html#kernels>



# Jupyter tabs

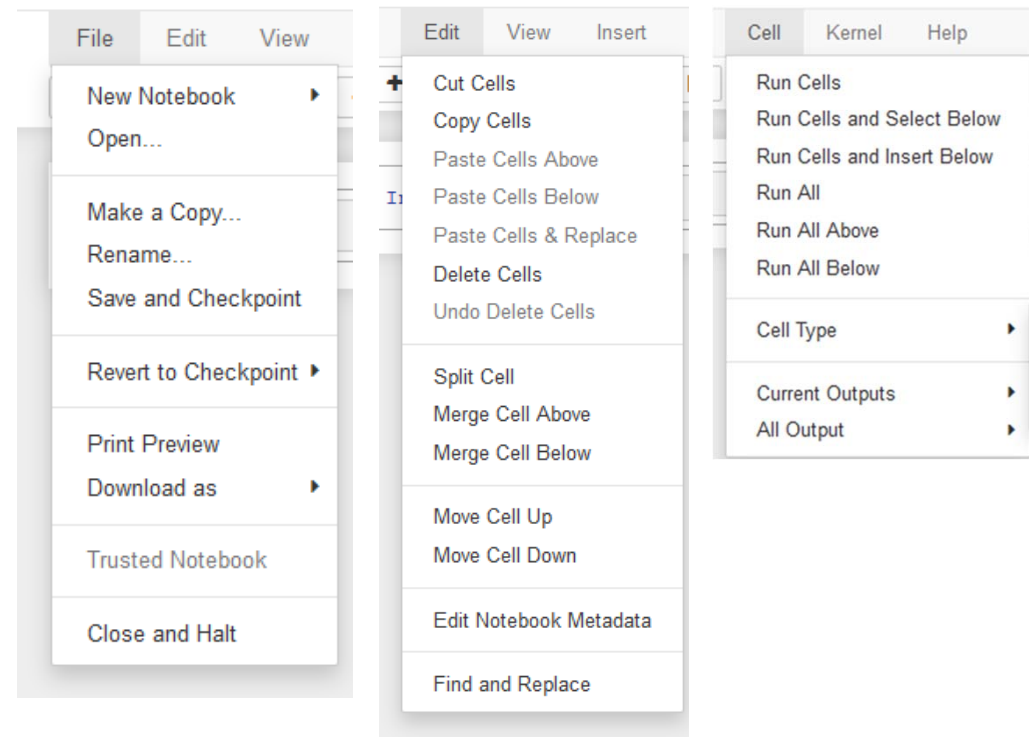
- File explorer
- Running kernels
  - Avoid working with multiple running kernels at the same time
  - Make sure kernel is stop before stop Jupyter server
- Clusters for parallel processing





# Jupyter Notebook's menu

- File: Standard file operations
- Edit: For editing cell contents (more to come)
- View: To change the display of the notebook
- Insert: To insert a cell in the notebook
- Cell: To change the format, usage of a cell
- Kernel: To adjust the kernel used for the notebook
- Help: To bring up the help system for Jupyter



# Jupyter Shortcuts

<https://www.cheatography.com/weidadeyue/cheat-sheets/jupyter-notebook/>

Enter	enter edit mode
Shift-Enter	run cell, select below
Ctrl-Enter	run cell
Alt-Enter	run cell, insert below
Y	to code
M	to markdown
R	to raw
1	to heading 1
2,3,4,5,6	to heading 2,3,4,5,6

# What is markdown?

- Lightweight language
- Plain text formatting syntax
- It's easily converted to HTML
- Is often used to format README files
- Try online editor: <https://stackedit.io>

# StackEdit

stackedit.io/app#

⊕ ☆ ⓧ 📄 ⬇️ ⚙️ 👤 ⌵

📁 ↶ ↷ B I ⏏ 🔗 ☰ ☷ ⌵ ⌵ 🔗 📄

Welcome file

#

# Welcome to MARKDOWN!

This is the main paragraph

## Create subsections

First Subsection including some **bold** and some *italic* terms

Or using bullet points:

- \* like this
- \* or this

|

Or including some code:

```
> variable1="uno"
> variable2="dos"
```

📄

📄

👁

Welcome to MARKDOWN!

This is the main paragraph

Create subsections

First Subsection including some **bold** and some *italic* terms

Or using bullet points:

- like this
- or this

Or including some code:

```
variable1="uno"
variable2="dos"
```

🔍

⬆️ ⬆️

📄



Anaconda Development Platform

# Variables, Expressions & Operators

Data Types

Conditionals

Loops

Functions

Files

## Variables and Types

```
In [2]: ► a=1  
        type(a)
```

```
Out[2]: int
```

```
In [3]: ► b="uno"  
        type(b)
```

```
Out[3]: str
```

```
In [4]: ► c="1"  
        type(c)
```

```
Out[4]: str
```

---

# Variable types

- There are some basic types
    - int
    - float
    - complex
    - boolean
    - str
  - Sequence types
    - tuple
    - list
    - range
  - Mapping types
    - dict
  - Iterator types
  - Binary types
    - bytes
    - bytearray
  - Set types
    - set
    - frozenset
-

# Conversions

Conversion between types is commonly required

- to string → `str()`
- to int → `int()`
- to float → `float()`

```
## This line will fail
c=12.12
print("Strings can only be concatenated with strings"+c)
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-17-aee039a69549> in <module>
      1 ## This line will fail
      2 c=12.12
----> 3 print("Strings can only be concatenated with strings"+c)

TypeError: can only concatenate str (not "float") to str
```

```
## This line will fail
c=12.12
e="15.56"
print("Numeric values can only be operated with numeric values")
c+e
```

Numeric values can only be operated with numeric values

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-18-e563bd88cf57> in <module>
      3 e="15.56"
      4 print("Numeric values can only be operated with numeric values")
----> 5 c+e

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



# Naming rules

```
In [6]: break = "Good morning"
```

```
File "<ipython-input-6-c8d24f2c986b>", line 1
    break = "Good morning"
    ^
SyntaxError: invalid syntax
```

```
In [7]: 12variable = 12.0
```

```
File "<ipython-input-7-d6a34504f326>", line 1
    12variable = 12.0
    ^
SyntaxError: invalid syntax
```

```
In [8]: savings_in_€ = 1200.00
```

```
File "<ipython-input-8-23587e84af81>", line 1
    savings_in_€ = 1200.00
    ^
SyntaxError: invalid character in identifier
```

# Arithmetic operators

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	$a + b = 30$
- Subtraction	Subtracts right hand operand from left hand operand.	$a - b = -10$
* Multiplication	Multiplies values on either side of the operator	$a * b = 200$
/ Division	Divides left hand operand by right hand operand	$b / a = 2$
% Modulus	Divides left hand operand by right hand operand and returns remainder	$b \% a = 0$
** Exponent	Performs exponential (power) calculation on operators	$a ** b = 10 \text{ to the power } 20$
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) –	$9 // 2 = 4$ and $9.0 // 2.0 = 4.0$ , $-11 // 3 = -4$ , $-11.0 // 3 = -4.0$

# Comparison operators

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

# Bitwise operators

Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
Binary OR	It copies a bit if it exists in either operand.	(a   b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

# Logical operators

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

# Membership operators

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

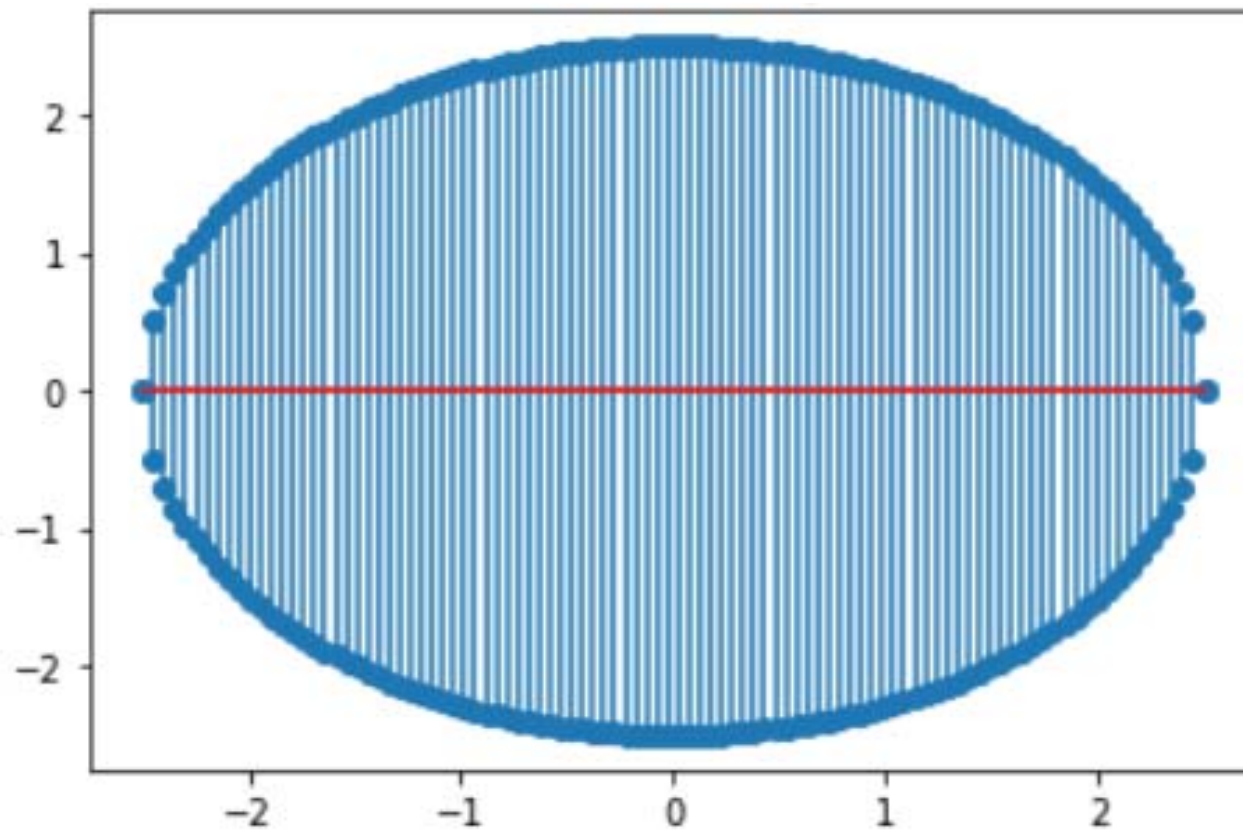
Operator	Description
**	Exponentiation (raise to the power)
~ + -	Ccomplement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

# LAB: Print circle with parameters

1. Ask the user for a radius:  
`radius=input("Write a radius and hit ENTER: ")`
  2. Create the x axis values using the NUMPY linspace equation from  $-radius$  to  $+radius$   
<https://docs.scipy.org/doc/numpy/reference/generated/numpy.linspace.html>
  3. Compute "y1" variable as  $y1 = \sqrt{radius^2 - x^2}$
  4. Compute "y2" variable as  $y2 = -\sqrt{radius^2 - x^2}$
  6. Plot the positive part of the circle using the MATPLOTLIB method `plt.stem()`  
([https://matplotlib.org/3.1.1/api/\\_as\\_gen/matplotlib.pyplot.stem.html](https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.stem.html))
  7. Plot the negative part of the circle
  8. Set a title using the MATPLOTLIB method `plt.title()`
-



## LAB: Print circle with parameters





Anaconda Development Platform  
Variables, Expressions & Operators

# Data Types

Conditionals

Loops

Functions

Files

# Numeric Types

```
In [16]: ▶ a=-10
          print(type(a))
          b=-10.10
          print(type(b))
          c=a+b
          print("value: {} of type: {}".format(c,type(c)))
          d=complex(2,2)
          e=d+a
          print("value: {} of type: {}".format(e,type(e)))

<class 'int'>
<class 'float'>
value: -20.1 of type: <class 'float'>
value: (-8+2j) of type: <class 'complex'>
```

# Booleans

```
a=10
b=20
print("a==b",a==b)
print("a>=b",a>=b)
print("a<=b",a<=b)
print("a!=b",a!=b)
```

```
a==b False
a>=b False
a<=b True
a!=b True
```

```
c=[1,2,3,4,5,6,7,8,9,10]
d=7
e=17
print("d is in list?",d in c)
print("e is in list?",e in c)
```

```
d is in list? True
e is in list? False
```

# Strings

```
a="This is the begining of the sentence"  
b="and this is the end of the sentence"  
print(a+" "+b)
```

This is the begining of the sentence and this is the end of the sentence

```
a="watermelon"  
b="waterloo"  
c="waterworld"  
print("watermelon larger than waterloo:",a>b)  
print("watermelon smaller than waterworld:",a<c)
```

watermelon larger than waterloo: True  
watermelon smaller than waterworld: True

---

- Finding

```
log="01/01/2019 18:45:00 MSG: Reboot of main controller SEVERITY: Heavy ACTION: Shutdown"  
start=log.find("MSG")  
end=log.find("SEVERITY")  
print(log[start:end])
```

MSG: Reboot of main controller

- Changing case

```
print(log.upper())  
print(log.lower())
```

01/01/2019 18:45:00 MSG: REBOOT OF MAIN CONTROLLER SEVERITY: HEAVY ACTION: SHUTDOWN  
01/01/2019 18:45:00 msg: reboot of main controller severity: heavy action: shutdown

---

# Split

```
In [1]: ▶ line="Agueda|25|Madrid|Student|Single|A"
        elements=line.split("|")
        print(elements)
        name=elements[0]
        age=elements[1]
        city=elements[2]
        job=elements[3]
        martial=elements[4]
        driving=elements[5]
        print("{0} {1} years old lives in {2} as {3} with driving licence {4}".format(name,age,city,job,driving))

['Agueda', '25', 'Madrid', 'Student', 'Single', 'A']
Agueda 25 years old lives in Madrid as Student with driving licence A
```

## Join

```
In [2]: ► list1=["Madrid","Berlin","London","Paris","Rome"]  
print(type(list1))  
print(", ".join(list1))  
print(" - ".join(list1))  
print(" * ".join(list1))
```

```
<class 'list'>
```

```
Madrid,Berlin,London,Paris,Rome
```

```
Madrid - Berlin - London - Paris - Rome
```

```
Madrid * Berlin * London * Paris * Rome
```



# Lists

```
number_list= [10,20,30,40,-50,-60,-80]  
print(type(number_list))
```

```
<class 'list'>
```

```
string_list= ["Madrid","Berlin","London","Paris","Rome"]  
print(type(string_list))
```

```
<class 'list'>
```

```
misc_list= ["monday", 35, True, complex(3,4),["Madrid","Berlin"]]  
print(type(misc_list))
```

```
<class 'list'>
```

```
empty=[]  
print(type(empty))
```

```
<class 'list'>
```

---

```
In [4]: ► sequence=["zero","one","two","three"]
print(sequence[0])
print(sequence[1])
print(sequence[2])
print(sequence[3])
print(sequence[-1])
print(sequence[-2])
print(sequence[-3])
print(sequence[-4])
print(sequence[4])
```

```
zero
one
two
three
three
two
one
zero
```

```
-----
IndexError                                Traceback (most recent call last)
<ipython-input-4-7d33216fd1ab> in <module>
      8 print(sequence[-3])
      9 print(sequence[-4])
--> 10 print(sequence[4])
```

```
IndexError: list index out of range
```

```
In [9]: ▶ sequence=["banana","apple","watermelon","strawberry"]  
a="pineapple" in sequence  
print("Pineapple in list: ",a)  
b="watermelon" in sequence  
print("Watermelon in list: ",b)
```

```
Pineapple in list: False  
Watermelon in list: True
```

```
In [10]: ▶ sequence=["zero","one","two","three"]  
         for number in sequence:  
             print("Element: ",number)
```

```
Element: zero  
Element: one  
Element: two  
Element: three
```

Like strings, lists are not numeric, but we can use some arithmetic operators to concatenate or to replicate

```
In [11]: ▶ sequence1=["banana","apple","watermelon","strawberry"]  
sequence2=["zero","one","two","three"]  
sequence1+sequence2
```

```
Out[11]: ['banana', 'apple', 'watermelon', 'strawberry', 'zero', 'one', 'two', 'three']
```

```
In [18]: ▶ sequence2*2
```

```
Out[18]: ['zero', 'one', 'two', 'three', 'zero', 'one', 'two', 'three']
```

Lists can be sliced

```
In [23]: ▶ sequence=["zero","one","two","three","four","five"]  
print(sequence[:3])  
print(sequence[2:5])  
print(sequence[4:])  
print(sequence[-3:])
```

```
['zero', 'one', 'two']  
['two', 'three', 'four']  
['four', 'five']  
['three', 'four', 'five']
```

List can contain Lists, that's to say "NESTED LIST"

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  
print(matrix)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
row=matrix[1]  
print(row)
```

```
[4, 5, 6]
```

```
element=row[1]  
print(element)  
element2=matrix[1][1]  
print(element2)
```

```
5
```

```
5
```

---

# Lists Methods

- `list.append(element)` → add “element” at the end of the list
- `list.sort()` → sort items using “<” relationship
- `list.extend([element1, ...,elementN])` → add items to list
- `list.reverse()` → reverse list
- `list.insert(position,element)` → insert element in position in list
- `list.remove(element)` → remove element from list



# Lists Methods

- `list.count(element)` → return the number of occurrences of element in list
- `list.pop()` → delete and return the last element
- `list.index(element)` → return the index of the element

# Tuples

```
tuple1=1,2,3,4
print("This is a tuple: ",tuple1)
print(type(tuple1))
tuple2=("four",5,"six",7)
print("This is also a tuple: ",tuple2)
print(type(tuple2))
```

```
This is a tuple: (1, 2, 3, 4)
<class 'tuple'>
This is also a tuple: ('four', 5, 'six', 7)
<class 'tuple'>
```

```
print("they can be indexed: ",tuple1[3])
print("they can be sliced: ",tuple1[:3])
```

```
they can be indexed: 4
they can be sliced: (1, 2, 3)
```

# Dictionaries

```
dictionary={"hola":"bonjour","adios":"aurevoir","hasta pronto":"a bientot"}  
print(dictionary)
```

```
{'hola': 'bonjour', 'adios': 'aurevoir', 'hasta pronto': 'a bientot'}
```

```
print("Hola in french: ",dictionary["hola"])  
print("Adios in french: ",dictionary["adios"])  
print("Hasta pronto in french: ",dictionary["hasta pronto"])
```

```
Hola in french:  bonjour
```

```
Adios in french:  aurevoir
```

```
Hasta pronto in french:  a bientot
```

---

## Easy to create [\(Chapter 8 from Learning Python 5<sup>th</sup> Edition\)](#)

```
dictionary={"hola":"bonjour","adios":"aurevoir","hasta pronto":"a bientot"}  
print(dictionary)
```

```
{'hola': 'bonjour', 'adios': 'aurevoir', 'hasta pronto': 'a bientot'}
```

```
dictionary={}  
dictionary["patata"]="pomme"  
dictionary["cinco"]=5  
dictionary["naranja"]="orange"  
dictionary["bolsa"]=["bourse","sac"]  
print(dictionary)
```

```
{'patata': 'pomme', 'cinco': 5, 'naranja': 'orange', 'bolsa': ['bourse', 'sac']}
```

---

## Easy to update

```
dictionary={"Teo":"admin","Liam":"viewer","Dora":"root","Noa":"viewer","Cris":"Viewer"}
print(dictionary)
dictionary["Noa"]="admin"
print(dictionary)
```

```
{'Teo': 'admin', 'Liam': 'viewer', 'Dora': 'root', 'Noa': 'viewer', 'Cris': 'Viewer'}
{'Teo': 'admin', 'Liam': 'viewer', 'Dora': 'root', 'Noa': 'admin', 'Cris': 'Viewer'}
```

```
print(dictionary)
print("Length: ",len(dictionary))
del dictionary["Teo"]
print(dictionary)
print("Length: ",len(dictionary))
```

```
{'Teo': 'admin', 'Liam': 'viewer', 'Dora': 'root', 'Noa': 'admin', 'Cris': 'Viewer'}
Length: 5
{'Liam': 'viewer', 'Dora': 'root', 'Noa': 'admin', 'Cris': 'Viewer'}
Length: 4
```

As lists, we can also nest dictionaries

```
registry={"Name":"Bob",  
         "Job":["Developer","Scrum master"],  
         "Web":"linkedin.com/bob",  
         "Personal":{  
             "Nationality":"British",  
             "Age":36,  
             "Status":"Divorced",  
             "Address":{  
                 "Street":"Fake Street",  
                 "Number":123  
             }  
         }  
     }}
```

```
print(registry["Name"])  
print(registry["Personal"]["Nationality"])  
print(registry["Personal"]["Address"]["Street"])
```

```
Bob  
British  
Fake Street
```

# LAB: Hands on lists

1. Create a variable that contains a list with different element types and length of your choice
  2. Test slicing using `[:start]`, `[start, end]` and `[:end]` as in the theory session
  3. Test every single method from the next two slides
-

# LAB: Hands on lists

- `list.append(element)` → add “element” at the end of the list
- `list.sort()` → sort items using “<” relationship
- `list.extend([element1, ...,elementN])` → add items to list
- `list.reverse()` → reverse list
- `list.insert(position,element)`
- `list.remove(element)`



# LAB: Hands on lists

- `list.count(element)` → return the number of occurrences of element in list
- `list.pop()` → delete and return the last element
- `list.index(element)` → return the index of the element



Anaconda Development Platform  
Variables, Expressions & Operators

Data Types

# Conditionals

Loops

Functions

Files

# Conditional execution

```
In [1]: ▶ x=10
        if x > 0:
            print("x is positive")
        else:
            print("x is negative")
```

x is positive

But...

What happens when  $x = 0$  ??

```
In [2]: ▶ x=-10
        if x > 0:
            print("x is positive")
        else:
            print("x is negative")
```

x is negative

Conditions can be chained using “**elif condition:**” meaning “**else: if condition:**”

```
In [5]: ▶ x=0
        if x > 0:
            print("x is positive")
        elif x < 0:
            print("x is negative")
        else:
            print("x is zero")
```

```
x is zero
```

In [8]: ▶

```
x=25
if x > 50:
    print("x is larger than 50")
elif x > 20:
    print("x is larger than 20")
elif x > 10:
    print("x is larger than 10")
else:
    print("x is fewer than 10")
```

x is larger than 20

Avoid nesting if possible

```
x=25
if x>0:
    if x>=50:
        print("x is larger or equal than 50")
    else:
        print("x is fewer than 50 but larger than 0")
elif x<0:
    if x <=-50:
        print("x is fewer or equal than -50")
    else:
        print("x is larger -50 but fewer than 0")
else:
    print("x=0")
```

x is fewer than 50 but larger than 0

Vs

Avoid nesting if possible

Vs

```
x=25
if x>=50:    print("x is larger or equal than 50")
elif x>0:    print("x is fewer than 50 but larger than 0")
elif x==0:   print("x=0")
elif x<=-50: print("x is fewer or equal than -50")
else:        print("x is larger -50 but fewer than 0")
```

x is fewer than 50 but larger than 0

---

Chained and nested conditionals can become simpler and easier to read using logical operators

```
In [16]: ▶ x=5  
if 0 < x:  
    if x < 10:  
        print("x is a positive single digit.")  
  
x is a positive single digit.
```

```
In [17]: ▶ x=5  
if 0 < x and x < 10:  
    print("x is a positive single digit.")  
  
x is a positive single digit.
```



- Logical operators stop evaluating (“short circuit”) as soon as a result is known
- This is taken into account on performance programming

**TRUE** or (**TRUE** and (**FALSE** and not(**FALSE** and **TRUE**)))



Stop evaluation here

Depending on the indent the “else” block means “y<=0” or “x<=0”

```
x = 1
print('this is block 0')
if x > 0:
    print("this is main branch of block 1")
    y = 1
    if y > 0:
        print('this is main branch of block 2')
    else:
        print('this is else branch of block 2')
    print('this is still main branch of block 1')
print('this is block 0, again')
```

```
x = 1
print('this is block 0')
if x > 0:
    print("this is main branch of block 1")
    y = 1
    if y > 0:
        print('this is main branch of block 2')
    else:
        print('this is else branch of block 1')
        print('this is still else branch of block 1')
print('this is block 0, again')
```

If the conditions are not very long and the execution block is not very long either, everything can be written on the same line

```
In [35]:  ▶ x=25
          if x>=50:    print("x is larger or equal than 50")
          elif x>0:    print("x is fewer than 50 but larger than 0")
          elif x==0:    print("x=0")
          elif x<=-50:  print("x is fewer or equal than -50")
          else:         print("x is larger -50 but fewer than 0")

          x is fewer than 50 but larger than 0
```

- Any nonzero number or nonempty object is true
- Zero numbers, empty objects, and the special object None are considered false

```
x = True
if x:
    print("condition satisfied")
```

condition satisfied

```
x = 5
if x:
    print("condition satisfied")
```

condition satisfied

```
x = {}
if x:
    print("condition satisfied")
else:
    print("condition false")
```

condition false

---

# Example 1: Membership verification

```
sentence="Mississippi is the second longest river in the United States"
letterCounts={}
for letter in sentence:
    if letter in letterCounts:
        letterCounts[letter]=letterCounts[letter]+1
    else:
        letterCounts[letter]=1
print(letterCounts)
```

```
{'M': 1, 'i': 8, 's': 8, 'p': 2, ' ': 9, 't': 6, 'h': 2, 'e': 7, 'c': 1,
'o': 2, 'n': 4, 'd': 2, 'l': 1, 'g': 1, 'r': 2, 'v': 1, 'U': 1, 'S': 1,
'a': 1}
```

## Example 2: Multiple choice

```
In [ ]: ▶ print("Please, select an action to do with your file")
        print("- Delete")
        print("- Rename")
        print("- Clone")
        print("- Move")
        action = input()
        if action == "Delete":
            print("Call delete function")
        elif action == "Rename":
            print("Call rename function")
        elif action == "Clone":
            print("Call clone function")
        elif action == "Move":
            print("Call move function")
        else:
            print("Action unkown")
```

## Example 3: Rules checking

```
vehicles=dict(car1=dict(model="saab93",length=464, width=176, height=145, weight=1570),
              car2=dict(model="fordfocus",length=439, width=182, height=148, weight=1493),
              car3=dict(model="nissanevalia",length=440, width=169, height=186, weight=1431),
              car4=dict(model="bmwq7",length=515, width=200, height=180, weight=2445),
              car5=dict(model="citroends3",length=395, width=171, height=148, weight=1090))
```

```
max_length=600
max_width=200
max_height=180
max_weight=2000
for index in vehicles:
    car=vehicles[index]
    if (car["length"]<max_length and
        car["width"]<max_width and
        car["height"]<max_height and
        car["weight"]<max_weight):
        print("Car ",car["model"]," can park")
    else:
        print("Car ",car["model"]," cannot park")
```

## Example 3: Avoid exceptions

```
import numpy as np
elements=[100,100,-100,-100,50,-50]
mean=np.mean(elements)
std=np.std(elements)
print("Average: ",mean)
print("Standard deviation: ",std)
```

Average: 0.0  
Standard deviation: 86.60254037844386

```
std_relative_to_mean=std/mean
```

C:\Users\luis\Anaconda3\lib\site-packages\ipykernel\_launcher.py:1: RuntimeWarning: divide by zero encountered in double\_scalars

"""Entry point for launching an IPython kernel.

Vs

```
if mean!=0:
    std_relative_to_mean=std/mean
else:
    std_relative_to_mean=np.inf
print("STD relative to average: ",std_relative_to_mean)
```

STD relative to average: inf



# LAB: Password Validation

1. Ask the user to introduce a password
  2. The password must fits some rules:
    - Contain 8 characters
    - Contain at least 1 letter lower case
    - Contain at least 1 letter upper case
    - Contain at least 1 digit
    - Contain at least 1 symbol
  3. Print a success message when the password is correct
-



Anaconda Development Platform  
Variables, Expressions & Operators

Data Types  
Conditionals

**Loops**

Functions  
Files

## Example 1: Process elements in a collection (list, dictionaries, tuples,...)

```
In [23]: ► sentence="Mississippi is the second longest river in the United States"
letterCounts={}
for letter in sentence:
    letterCounts[letter] = letterCounts.get (letter, 0) + 1
print(letterCounts)

{'M': 1, 'i': 8, 's': 8, 'p': 2, ' ': 9, 't': 6, 'h': 2, 'e': 7, 'c': 1, 'o': 2, 'n': 4, 'd': 2,
'l': 1, 'g': 1, 'r': 2, 'v': 1, 'U': 1, 'S': 1, 'a': 1}
```

## Example 2: Attempts on an user input request

```
In [1]: ▶ for i in range(1000):  
        login=input("Chose a 8 characters user name: ")  
        if len(login)==8:  
            break
```

Chose a 8 characters user name: Luis

Chose a 8 characters user name: Luis

Chose a 8 characters user name: LuisLuis

## Example 3: Hyper-parameter selection

```
In [ ]: ▶ X=data
Y=labels
all_classifiers=[]
all_accuracy=[]

for n in [10,50,100,150,200]:
    for d in [1,2,3,4,5,6,7,8,9,10]:
        for f in ["auto","sqrt","log2"]:

            classifier = RandomForestClassifier(n_estimators=n, max_depth=d, max_features=f )
            accuracy = classifier.fit(X, y)
            all_classifiers.append(classifier)
            all_accuracy.append(accuracy)

best_accuracy=max(all_accuracy)
index = all_accuracy.index(best_accuracy)
best_classifier = all_classifiers[index]
```

# For Loop

```
for target in object:  
    statements  
  
else:  
    statements
```

```
In [25]: ▶ print("From 0 to 10 with step=2")  
for i in range(0,10,2):  
    print(i)
```

```
From 0 to 10 with step=2
```

```
0
```

```
2
```

```
4
```

```
6
```

```
8
```

---

```
In [28]: ► import numpy as np  
print("Range of 5 elements from 0 to 10")  
for i in np.linspace(0,10,5):  
    print(i)
```

Range of 5 elements from 0 to 10

0.0

2.5

5.0

7.5

10.0



```
In [2]: ▶ prod=1  
for i in [1,2,3,4,5,6]:  
    prod *= i  
    print(prod)
```

1

2

6

24

120

720

```
In [5]: ► for letter in "Kinesiotherapy":  
          print(letter,end='_')
```

```
K_i_n_e_s_i_o_t_h_e_r_a_p_y_
```

```
In [6]: ► list_of_tuples=[("uno",1),("dos",2),("tres",3),("cuatro",4),("cinco",5)]  
        for k,v in list_of_tuples:  
            print(k," is ",v)
```

```
uno is 1  
dos is 2  
tres is 3  
cuatro is 4  
cinco is 5
```

```
In [8]: ► dictionary={"Teo":"admin","Liam":"viewer","Dora":"root","Noa":"viewer","Cris":"Viewer"}  
        for k,v in dictionary.items():  
            print(k," has the rol of ",v)
```

```
Teo  has the rol of  admin  
Liam has the rol of  viewer  
Dora has the rol of  root  
Noa  has the rol of  viewer  
Cris has the rol of  Viewer
```

```
In [16]: ▶ sequence = [("Leo",("Maths","Literature")),  
                        ("Ariel",("Art","Etics")),  
                        ("Liu Xiao",("Algorithms","Programming")),  
                        ("Nico",("Music","Antropology"))]  
  
for (student,(subject1,subject2)) in sequence:  
    print(student," is registered in: ",subject1," and ",subject2)
```

```
Leo is registered in: Maths and Literature  
Ariel is registered in: Art and Etics  
Liu Xiao is registered in: Algorithms and Programming  
Nico is registered in: Music and Antropology
```

```
In [24]: ▶ import numpy as np
array = np.array([(1,2,3),(4,5,6),(6,7,8)])
print(array)
print("Numpy arrays can be accessed as A[row,col]")
rows,cols=array.shape
for r in range(rows):
    for c in range(cols):
        print("Array element [{},{}]".format(r,c)," value: ",array[r,c])
```

```
[[1 2 3]
 [4 5 6]
 [6 7 8]]
```

```
Numpy arrays can be accessed as A[row,col]
```

```
Array element [0,0] value: 1
```

```
Array element [0,1] value: 2
```

```
Array element [0,2] value: 3
```

```
Array element [1,0] value: 4
```

```
Array element [1,1] value: 5
```

```
Array element [1,2] value: 6
```

```
Array element [2,0] value: 6
```

```
Array element [2,1] value: 7
```

```
Array element [2,2] value: 8
```

# While loop

```
while test:  
    statements  
else:  
    statements
```

---

```
In [1]: ► stay=True
while stay==True:
    command=input("Write exit to abort: ")
    if command=="abort": stay=False
print("EXIT!")
```

```
Write exit to abort: Ausgang
Write exit to abort: Exit
Write exit to abort: Salir
Write exit to abort: Sortir
Write exit to abort: abort
EXIT!
```

This can be condensed to:

```
while stay:
    command = input()
```



Beware of infinite loops!!

```
In [4]: ► x= "Kinesiotherapy"  
while x:  
    print(x)
```

```
Kinesiotherapy  
Kinesiotherapy  
Kinesiotherapy  
Kinesiotherapy  
Kinesiotherapy  
Kinesiotherapy  
Kinesiotherapy  
Kinesiotherapy  
Kinesiotherapy  
Kinesiotherapy  
Kinesiotherapy  
...
```

Instead of iterating between all characters, this CONDITION will be evaluated always as TRUE

In [5]: ▶

```
x= "Kinesiotherapy"  
while x:  
    print(x)  
    x=x[1:]
```

```
Kinesiotherapy  
inesiotherapy  
nesiotherapy  
esiotherapy  
siotherapy  
iotherapy  
otherapy  
therapy  
herapy  
erapy  
rapy  
apy  
py  
y
```

This is the most common use case

```
stocks = 1000
stock_price = 10
losts = 0
threshold = 1000

print("I payed ",stocks*stock_price, "€ to buy ", stocks, " stocks")
print("I accept to lose 1000€ maximum")

while losts < threshold:
    new_price = np.random.normal(10, 1, 1)[0]
    print("New day, stock price: ",new_price)
    losts = 10000 - stocks*new_price
    if losts >=0: print("Losts: ",losts)
    else: print("Gains: ",-losts)
```

Note that the number of interactions is unknown

```
I payed 10000 € to buy 1000 stocks
I accept to lose 1000€ maximum
New day, stock price: 9.724739084418989
Losts: 275.26091558101143
New day, stock price: 10.234306023571058
Gains: 234.30602357105818
New day, stock price: 12.017374196377578
Gains: 2017.3741963775774
New day, stock price: 12.089287044826763
Gains: 2089.287044826764
New day, stock price: 10.426780131052157
Gains: 426.78013105215723
New day, stock price: 8.725643468052738
Losts: 1274.3565319472618
```

“Break” is specially useful when requesting user inputs

```
In [1]: ► while True:
        name = input('Enter name:')
        if name == 'stop': break
        age = input('Enter age: ')
        print('Hello', name, ', this is your age squared =>', int(age) ** 2)
```

```
Enter name:Luis
```

```
Enter age: 41
```

```
Hello Luis , this is your age squared => 1681
```

```
Enter name:stop
```

Note that many loops can be implemented with while and with for statements

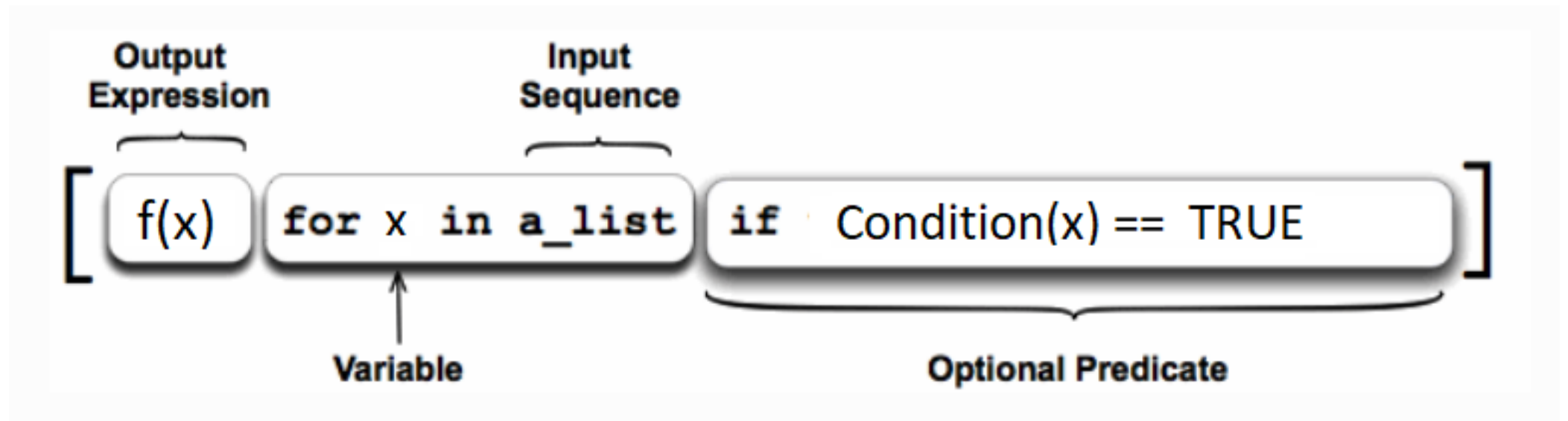
```
In [50]: ► x = 1
          while x:
              x=x+1
              if x % 2 != 0: continue
              print(x, end=' ')
              if x == 100: break
```

```
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 4
0 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 7
6 78 80 82 84 86 88 90 92 94 96 98 100
```

```
In [52]: ► for x in range(1,100,1):
          if x % 2 != 0: continue
          print(x, end=' ')
```

```
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 4
0 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 7
6 78 80 82 84 86 88 90 92 94 96 98
```

# List Comprehensions



# Easy

We apply a transformation to every element

We can use comprehensions on STRINGS

```
In [10]: ► [x.upper() for x in "change case"]
```

```
Out[10]: ['C', 'H', 'A', 'N', 'G', 'E', ' ', 'C', 'A', 'S', 'E']
```

```
In [12]: ► [x.upper() for x in "n0w w1th numb3rs" if x.islower()]
```

```
Out[12]: ['N', 'W', 'W', 'T', 'H', 'N', 'U', 'M', 'B', 'R', 'S']
```

Apply transformation only to those elements  
that match a condition

# Easy

```
sequence=('blowfish', 'clownfish', 'catfish', 'octopus')  
[x.upper() for x in sequence if "fish" in x]
```

```
['BLOWFISH', 'CLOWNFISH', 'CATFISH']
```

```
dictionary={"Hola":"spanish","Bonjour":"french","Hello":"english",  
            "Adios":"spanish","Au revoir":"french","Bye":"english"}  
[k for k,v in dictionary.items() if v=="spanish" ]
```

```
['Hola', 'Adios']
```



# Medium

```
my_list = []  
  
for x in [20, 40, 60]:  
    for y in [2, 4, 6]:  
        my_list.append(x * y)  
  
print(my_list)
```

```
[40, 80, 120, 80, 160, 240, 120, 240, 360]
```

```
my_list = [x * y for x in [20, 40, 60] for y in [2, 4, 6]]  
my_list
```

```
[40, 80, 120, 80, 160, 240, 120, 240, 360]
```

# Medium

```
list_of_list = [[1, 2, 3], [4, 5, 6], [7], [8, 9]]
```

```
flat_list = []  
for sublist in list_of_list:  
    for item in sublist:  
        flat_list.append(item)
```

```
flat_list
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
flat_list = [item for sublist in list_of_list for item in sublist]  
flat_list
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

# Medium

```
number_list=[]  
for x in range(100):  
    if x % 3 == 0 :  
        if x % 5 == 0 :  
            number_list.append(x)  
print("List of 3 and 5 multiples")  
print(number_list)
```

List of 3 and 5 multiples  
[0, 15, 30, 45, 60, 75, 90]

```
number_list = [x for x in range(100) if x % 3 == 0 and x % 5 == 0]  
print("List of 3 and 5 multiples")  
print(number_list)
```

List of 3 and 5 multiples  
[0, 15, 30, 45, 60, 75, 90]

---

# LAB: Password Generator

1. Ask the user for a password that fulfil the rules from the previous lab
  2. If the password does not fits those rules, print a message and ask again for a new password.
  3. The user have tree tries to input a valid password
  4. If the user exceeds his three tries, then the code must generate a random password that fits all the rules
-



Anaconda Development Platform  
Variables, Expressions & Operators

Data Types  
Conditionals

Loops

**Functions**

Files

A function is a “wrapper” of statements performing some actions

```
def function_name (arg1, arg2, arg3)
:
    ...
    ...
    ...
return value
```

---

- Function returning no values

```
In [29]: ► def func1(arg):  
           print("Just printing the argument:",arg)  
  
           out_func1=func1(100)  
           print(out_func1)  
  
           Just printing the argument: 100  
           None
```

- Function returning a value

```
In [31]: ► def func2(arg):  
           out= arg*2  
           return out  
  
           out_func2=func2(100)  
           print(out_func2)  
  
           200
```

All the values defined inside the function's body are local. You cannot use them outside

This is the scope of the function

```
def concat(string1,string2):  
    newstring= string1+" "+string2  
    return newstring
```

```
begining="This is the begining part"  
ending="and this is the ending part"  
sentence = concat(begining, ending)  
print(sentence)
```

This is the begining part and this is the ending part

```
print(newstring)
```

```
-----  
NameError                                Traceback (most recent call last)  
<ipython-input-34-a997591fe6ad> in <module>  
----> 1 print(newstring)
```

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



# Functions Scope

```
from datetime import datetime
def addTimeStamp():
    now = datetime.now()
    return "timestamp: " +str(now)

def checkIn(arg):
    log = arg + " is checking IN "+addTimeStamp()
    return log

def checkout(arg):
    log = arg + " is checking OUT| "+addTimeStamp()
    return log

def cardReader(name,action):
    if action=="IN":
        log=checkIn(name)
    else:
        log=checkOut(name)
    return log
```

Cannot concatenate a datetime type with an string type without conversion

We can use the same variable name in different functions because scopes are different

An outer function can invoke an inner function that uses the same variable name because the scopes are different

The most common structure is returning some variable created inside

Using that functions  
make the code more  
readable and reduces  
repeated statements

```
In [47]: ▶ import math
def area(radius):
    temp = math.pi * radius**2
    return temp
```

```
print("The area of circle with radius 2 is: ",area(2))
print("The area of circle with radius 3 is: ",area(3))
print("The area of circle with radius 4 is: ",area(4))
print("The area of circle with radius 5 is: ",area(5))
```

```
The area of circle with radius 2 is: 12.566370614359172
The area of circle with radius 3 is: 28.274333882308138
The area of circle with radius 4 is: 50.26548245743669
The area of circle with radius 5 is: 78.53981633974483
```

More than one return statement is allowed

```
In [49]: ► import math
def area(radius):
    if radius < 0:
        return -1
    elif radius == 0:
        return 0
    else :
        return math.pi * radius**2

print("The area of circle with radius -3 is: ",area(-3))
print("The area of circle with radius 0 is: ",area(0))
print("The area of circle with radius 1 is: ",area(1))
print("The area of circle with radius 2 is: ",area(2))
```

```
The area of circle with radius -3 is:  -1
The area of circle with radius 0 is:  0
The area of circle with radius 1 is:  3.141592653589793
The area of circle with radius 2 is:  12.566370614359172
```

Note that all the code after a return statements will never be executed

```
In [50]: import math
def area(radius):
    temp = math.pi * radius**2
    return temp
    perimiter = 2 * math.pi * radius
    print("The area of a circle with radius {} is: {}".format(radius,temp))
    print("The perimiter of a circle with radius {} is: {}".format(radius,perimiter))

print("The area of circle with radius 2 is: ",area(2))
print("The area of circle with radius 3 is: ",area(3))
print("The area of circle with radius 4 is: ",area(4))
print("The area of circle with radius 5 is: ",area(5))
```

```
The area of circle with radius 2 is: 12.566370614359172
The area of circle with radius 3 is: 28.274333882308138
The area of circle with radius 4 is: 50.26548245743669
The area of circle with radius 5 is: 78.53981633974483
```

This is dead code

Arguments are not mandatory

```
In [25]: ► from datetime import datetime

def addTimeStamp():
    now = datetime.now()
    return "timestamp: " +str(now)

addTimeStamp()
```

```
Out[25]: 'timestamp: 2019-08-27 11:07:09.153470'
```

```
In [26]: ► import math
def areaCircle(radius):
    area = math.pi * radius**2
    return area

areaCircle(5)
```

Out[26]: 78.53981633974483

```
In [27]: ► def concat(string1,string2):
    newstring= string1+" "+string2
    return newstring

concat("Good","morning")
```

Out[27]: 'Good morning'

We can assign default values

```
from datetime import datetime
import pytz

def currentTimeStamp(timezone='Europe/Madrid'):
    tz = pytz.timezone(timezone)
    now = datetime.now(tz)
    return "Timestamp in {}: {}".format(timezone, now)
```

```
currentTimeStamp()
```

```
'Timestamp in Europe/Madrid: 2019-08-27 11:18:21.558840+02:00'
```

```
currentTimeStamp("Israel")
```

```
'Timestamp in Israel: 2019-08-27 12:18:50.595880+03:00'
```

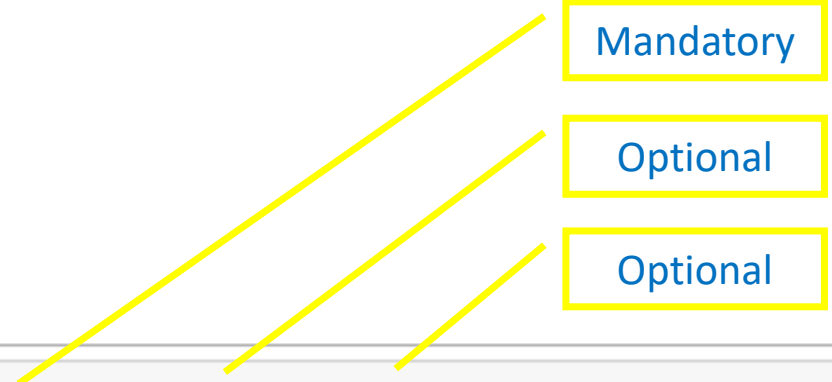
The arguments order is relevant

Mandatory

Optional

Optional

```
In [48]: ▶ def registerUser(name, city="Madrid", job="Student"):
           out="User {} from city: {} with job: {} created".format(name,city,job)
           return out
```





- How can we sort this issue out?

```
registerUser("Luis")
```

```
'User Luis from city: Madrid with job: Student created'
```

```
registerUser("Luis","Cuenca")
```

```
'User Luis from city: Cuenca with job: Student created'
```

```
registerUser("Luis","Cuenca","Enginner")
```

```
'User Luis from city: Cuenca with job: Enginner created'
```

```
registerUser("Luis","Enginner")
```

```
'User Luis from city: Enginner with job: Student created'
```

---

- Where there exists ambiguity, use explicit naming for the arguments

```
registerUser("Luis",job="Enginner")
```

```
'User Luis from city: Madrid with job: Enginner created'
```

- Note that Python is not TYPESAFE, so the type does not help with disambiguation

```
registerUser("Luis",100,200)
```

```
'User Luis from city: 100 with job: 200 created'
```

---

Sometimes the number of arguments is variable

```
def newUsers(*args):  
    print("Arguments type:", type(args))  
    print("Arguments: ", args)
```

```
newUsers()
```

```
Arguments type: <class 'tuple'>  
Arguments:  ()
```

```
newUsers("Mia", "Brian", "Roman", "Elena")
```

```
Arguments type: <class 'tuple'>  
Arguments:  ('Mia', 'Brian', 'Roman', 'Elena')
```

```
newUsers(1, 2, 3, 4, 5)
```

```
Arguments type: <class 'tuple'>  
Arguments:  (1, 2, 3, 4, 5)
```

---

More flexibility can be achieved using dictionaries

```
def newParkingPlaces(**args):  
    print("Arguments type:", type(args))  
    print("Arguments: ", args)
```

```
newParkingPlaces()
```

```
Arguments type: <class 'dict'>  
Arguments: {}
```

```
newParkingPlaces(engineering=5, it=10, sales=20, rrhh=5)
```

```
Arguments type: <class 'dict'>  
Arguments: {'engineering': 5, 'it': 10, 'sales': 20, 'rrhh': 5}
```

Or combine both solutions

```
def newBulding(city,*users,**parking):  
    print("City variable type:",type(city))  
    print("City: ",city)  
    print("Users variable type:",type(users))  
    print("Users: ",users)  
    print("Parking variable type:",type(parking))  
    print("Parking: ",parking)
```

```
newBulding("Madrid","Mia","Brian","Roman","Elena",engineering=5,it=10,sales=20,rrhh=5)
```

```
City variable type: <class 'str'>
```

```
City: Madrid
```

```
Users variable type: <class 'tuple'>
```

```
Users: ('Mia', 'Brian', 'Roman', 'Elena')
```

```
Parking variable type: <class 'dict'>
```

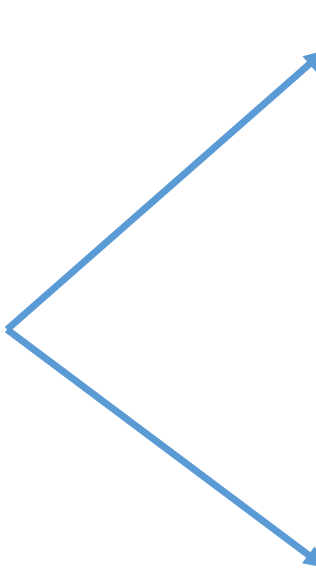
```
Parking: {'engineering': 5, 'it': 10, 'sales': 20, 'rrhh': 5}
```

# Anonymous functions

- Are small functions
  - They have no name
  - Can only have a single expression
  - Also known as LAMBDA functions
  - Very common on MAP operations
-

Note the differences

```
def sum2values(value1,value2):  
    return value1+value2
```



```
sum2values(10,20)
```

30

```
f=lambda value1,value2: value1+value2  
f(10,20)
```

30

---

Makes MAP operations straightforward. Mapping means applying the same transformation to all the elements of a collection

```
file=("Ann;18;Single","Leo;24;Single","Ron;55;Divorced")

def splitColumns(line):
    return line.split(";")

matrix=[]
for line in file:
    matrix.append(splitColumns(line))
```

```
for i in matrix: print(i)
```

```
['Ann', '18', 'Single']
['Leo', '24', 'Single']
['Ron', '55', 'Divorced']
```



```
file=("Ann;18;Single","Leo;24;Single","Ron;55;Divorced")

def splitColumns(line):
    return line.split(";")

matrix=[]
for line in file:
    matrix.append(splitColumns(line))
```



```
file=("Ann;18;Single","Leo;24;Single","Ron;55;Divorced")

def splitColumns(line):
    return line.split(";")

matrix = map( splitColumns , file )
```



This is the most common way of  
processing extremely large collections  
on Big Data infrastructures

```
file = ("Ann;18;Single","Leo;24;Single","Ron;55;Divorced")
matrix = map( lambda x:x.split(";") , file )
```

# LAB: Password Generator

1. Rewrite the code from previous Lab using these two functions:

```
def validate(password):  
  
    return boolean
```

In [ ]:

```
def generate_password():  
  
    return password
```

# LAB: Password Generator

2. Those functions must make this code work:

```
print("Now chose a password that fits:")
print("    - length: 8")
print("    - at least 1 lower case")
print("    - at least 1 upper case")
print("    - at least 1 digit")
print("    - at least 1 symbol")

for i in range(1,4,1):
    print("This is your try: ",i, ". You have ",3-i," remaining")
    password=input("Chose a 8 characters password: ")
    if validate(password):
        break
else:
    print("Your three attempts expired")
    password=generate_password()
print("This is your new password: ", password)
```



Anaconda Development Platform  
Variables, Expressions & Operators

Data Types  
Conditionals

Loops  
Functions

Files

"r"	Read - Default value. Opens a file for reading, error if the file does not exist	<i>#open file book.txt in reading mode:</i> <code>file1 = open('book.txt', 'r')</code>
"a"	Append - Opens a file for appending, creates the file if it does not exist	<code>file1 = open('book.txt', 'a')</code> <code>file1.write("\nhola de nuevo!")</code> <code>file1.close()</code>
"w"	Write - Opens a file for writing, creates the file if it does not exist	<i>#open file book.txt in writing mode:</i> <code>file1 = open('book.txt', 'w')</code> <code>file1.write("hola!")</code> <code>file1.close()</code>
+	Read and write	<code>file1 = open('book.txt', 'r+')</code> <code>print(file1.read())</code> <code>file1.write("\nNEW!")</code> <code>file1.close()</code>

WITH\_AS automatically close the file even if an exception is launched.

```
f1=open('temps.txt','r')  
print(f1.read())  
f1.close()
```

```
Madrid 25  
Badajoz 38  
Soria 15  
Segovia 23
```

VS

```
with open('temps.txt','r') as f1:  
    print(f1.read())
```

```
Madrid 25  
Badajoz 38  
Soria 15  
Segovia 23
```

---

print the whole file

```
▶ file1 = open('book.txt', 'r')  
print(file1.read())  
file1.close()
```

hola!

print a part of it

```
▶ file1 = open('book.txt', 'r')  
print(file1.read(3))  
file1.close()
```

hol

---

### print line by line

```
▶ file1 = open('book.txt', 'r')  
print(file1.readline())  
file1.close()
```

En un pais multicolor

### using loops

```
▶ file1 = open('book.txt', 'r')  
for x in file1:  
    print (x)  
file1.close()
```

En un pais multicolor

nacio una abeja bajo el sol

y fue famosa en el lugar



Function [read\(\)](#): reads from a file a string or the first *n* bytes of a string.

```
file1=open('book.txt','r')
str=file1.read()
print(len(str))
```

101

Function [readline\(\)](#): reads the file until the file pointer reaches the character end of line (\n)

```
file1=open('test.txt','r')
linea=file1.readline()
print(linea,len(linea))
linea=file1.readline()
print(linea,len(linea))
```

```
file
5
with several lines 18
```

Function [readlines\(\)](#): reads all strings of the files returning them like a list

```
file1=open('temps.txt','r')
lista=file1.readlines()
print(lista)
```

```
['Madrid 25\n', 'Badajoz 38\n', 'Soria 15\n', 'Segovia 23']
```

```
file1=open('temps.txt','r')
lista=file1.read().splitlines()
print(lista)
file1.close()
```

```
['Madrid 25', 'Badajoz 38', 'Soria 15', 'Segovia 23']
```

---

Function [write\(\)](#): writes a string into a file. Depends on and stream position

```
with open('cities.txt', 'w') as f1:  
    f1.write("Guadalajara\nMadrid\nSegovia\nSoria")
```

```
with open('cities.txt', 'r') as f1:  
    print(f1.read())
```

Guadalajara  
Madrid  
Segovia  
Soria

```
with open('cities.txt', 'a') as f1:  
    f1.write("\nSevilla")
```

```
with open('cities.txt', 'r') as f1:  
    print(f1.read())
```

Guadalajara  
Madrid  
Segovia  
Soria  
Sevilla

Function [writelines\(\)](#) : writes a group of strings such as a list of strings in a file, indicating the lines that we want to write them:

```
list=["Santander","Madrid","Toledo","Soria"]  
with open('cities2.txt','w') as f1:  
    f1.writelines(list)
```

```
with open('cities2.txt','r') as f1:  
    print(f1.read())
```

SantanderMadridToledoSoria

---

[tell\(\)](#): method returns the current file position in a file stream:

```
with open('cities.txt', 'r') as f1:  
    print(f1.read(17))  
    print(f1.tell())  
    print(f1.read(3))  
    print(f1.tell())
```

Guadalajara  
Madri  
18  
d  
S  
22

Next position to read will be 18! So its located at position 18.

seek(): method sets the current file position in a file stream

```
with open('cities.txt', 'r') as f1:  
    print(f1.read(18))  
    f1.seek(0)  
    print(f1.read(6))  
    f1.seek(21)  
    print(f1.read(7))
```

Guadalajara  
Madrid  
Guadal  
Segovia

---

# Binary Files

- We must know the format that data is written in order to read them
- To create a binary file we add the letter b to the mode of opening file.

```
with open('cities.dat', 'rb') as f1:
```

- Normally, binary file names have extension such as .bin, .dat, etc. .
-

Pickle library: this library allows to serialize python objects in binary files

<code>list=[]</code>	<b>#create a list of objects</b>
<code>#open file, f</code>	
<code>pickle.dump(list,f)</code>	<b>#write the list in file</b>
<code>new_list=pickle.load(f)</code>	<b>#reads file and creates a new_list</b>
<code>print(new_list)</code>	

---



**Dump method:** Allows to save (write) in a binary file different types of objects.

```
import pickle
file=open("icai.dat", "wb")
entero=2
real=4.7
cadena="Buenos dias"
lista=["Madrid", "Cordoba"]
tupla=(2, 3)
diccionario={"Algebra":7.3, "Calculo":7.8, "C":9.5}
pickle.dump(entero,file)
pickle.dump(real,file)
pickle.dump(cadena,file)
pickle.dump(lista,file)
pickle.dump(tupla,file)
pickle.dump(diccionario,file)
file.close()
```

load method: Allows to read in variables information from a file

```
file=open("icai.dat", "rb")
print("\nPrimera forma:")
for i in range(6):
    datos= pickle.load(file)
    print(datos)

file.seek(0)
datos=[]
print("\nSegunda forma:")
for i in range(6):
    datos.append(pickle.load(file))

print(datos)
```

# LAB: Count the frequency of words

1. Open the file “Dracula.txt”
2. Count the frequency of every single letter
3. Plot a graph like the one on this slide using the method:

```
import matplotlib.pyplot as plt
```

```
plt.bar(x,y)
```

