Conceptes bàsics de programació i Python, orientat a 1r de Batxillerat

Enric X. Martín Rull
Manel Velasco García
Juny 2022

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Estructura del curs:

PART 1. El llenguatge Python

PART 2. Conceptes d'algorísmia

PART 3. Utilització de llibreries

PART 4. Exercicis tipus

PART 1. El llenguatge Python

1. Objectiu, idees bàsiques
2. Dades: variables, tipus, constants
3. Operacions aritmètiques
4. Booleans, cadenes, conjunts i llistes
5. Interacció amb l'usuari
6. Fitxers, guardar dades
7. Funcions
8. Bifurcacions, repeticions
9. Comentaris

PART 2. Conceptes d'algorísmia

1. Conceptes d'algorísmia
2. Estructura del codi
3. Estructura de les dades
4. Creació de classes
5. Qüestions d'estil

PART 3. Utilització de llibreries

1. Utilització de llibreries
2. Creació de llibreries
3. Algunes llibreries d'interès

PART 4. Exercicis tipus

1. Programació d'un accés amb password
2. Endevinar un nombre
3. Comptar lletres
4. Daus i probabilitat
5. Tornar canvi
6. Encriptar i desencriptar
7. Cercar el mínim d'una paràbola
8. Primers i factors

Altres ítems del curs...

1. Plataformes, instal·lació
2. Didàctica, plantejament del curs
3. Motivació, gènere
4. Resolució de problemes i exercicis
5. Projectes, transversalitat
6. Organització del curs a Batxillerat

1. Python

PART 1. El llenguatge Python

1. Objectiu, idees bàsiques
2. Dades: variables, tipus, constants
3. Operacions aritmètiques
4. Booleans, cadenes, conjunts i llistes
5. Interacció amb l'usuari
6. Fitxers, guardar dades
7. Funcions
8. Bifurcacions, repeticions
9. Comentaris

1.1. Objectiu, idees bàsiques

Per què Python?

Avantatges i riscos

1.1. Objectiu, idees bàsiques

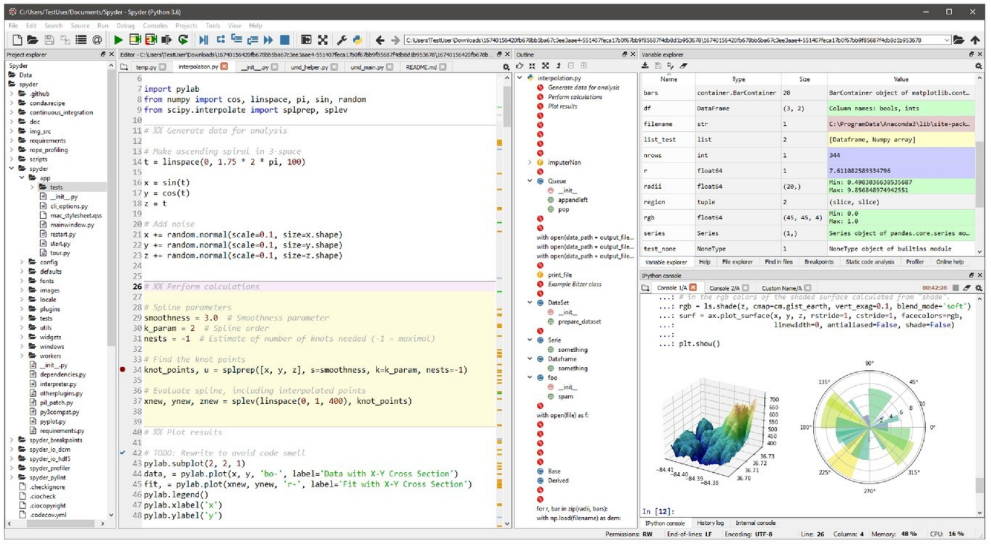
Per què Python?

- Lliure
- Aplicacions científiques
- Gran varietat de llibreries
- Comunitat: suport, exemples

- Varietat de plataformes i interfícies
- Llenguatge interpretat

- Llegibilitat del codi
- Feble al treball amb tipus

1.1. Objectiu, idees bàsiques



Plataformes de treball amb Python

- Pycharm (€)
- Pydev
- Geany

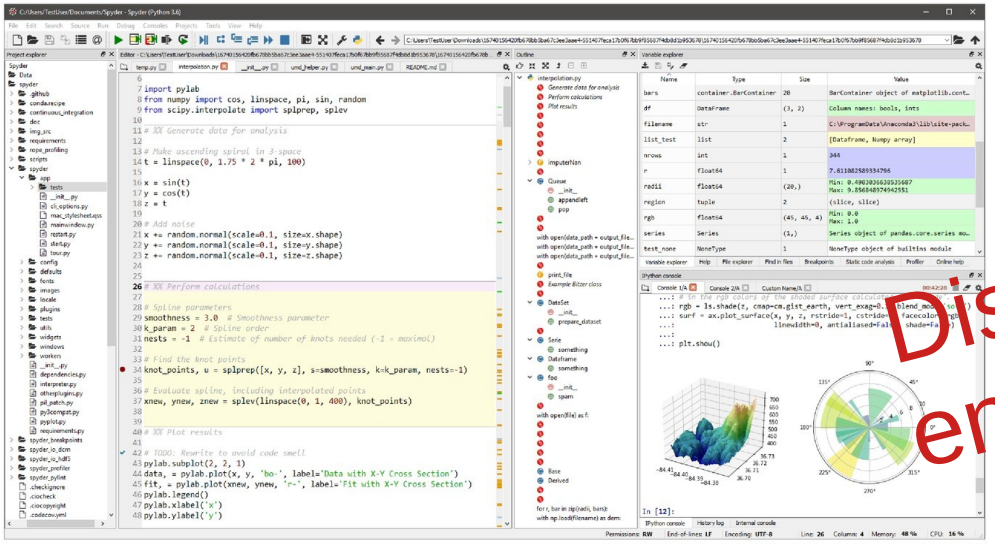
- **Spyder**

- Idle (a l'instal·lar Python a W)
  + editor qualsevol
- Visual Studio (W)

- PythonAnywhere (web)

- Solucions amb servidors

1.1. Objectiu, idees bàsiques



Discussió i posada en comú d'idees

Plataformes de treball amb Python

1.1. Objectiu, idees bàsiques

Conceptes de representació, calen? (doc)

Funcionament de màquines, cal? (prog)

Autonomia:

www.python.org

docs.python.org/3/reference

www.stackoverflow.com

1.2. Dades: variables, tipus, constants

Tipus	Exemple de creació	Descripció
enter (int)	x = 12	Variable per contenir nombres enters
real (float)	y = 4.2	Variable per contenir nombres reals
complexe (complex)	z = 5.0 + 2j	Variable per contenir nombres complexes (la lletra emprada és la j)
booleà (bool)	bit = False	Pot contenir estats lògics (True o False)
cadena (string)	s = 'Maria'	Conté seqüències de caràcters. Es poden definir amb cometa simple o doble (essent coherents a la definició)
tupla (tuple)	a = ('Pedro', 17)	Conté una estructura amb diversos camps, que poden tenir tipus diferents. Accedim amb als camps amb [ ] els camps es comencen a comptar per 0.

1.2. Dades: variables, tipus, constants

Tipus	Exemple de creació	Descripció
conjunt (set)	c = { 'blau', 'verd', 'vermell', 'groc' }	Variable per contenir elements no ordenats i no repetits i treballar amb operacions de conjunts.
rang (range)	r = range (-3, 3, 1)	Els rangs es defineixen amb un interval i un pas, es proporciona el valor inicial, el final i el pas. A l'exemple tindrem que el rang conté: -3, -2, -1, 0, 1 i 2. *Atenció, el "final" no entra al rang!*
llista (list)	m = [2, 4, 1, 7, 3]	Seqüència ordenada d'elements, a diferència dels conjunts pot haver elements repetits. Els elements de la llista poden ser heterogenis, inclús pot haver llistes dins de les llistes!
diccionari (dict)	d = { 1:'a', 2:'b', 3:'c' }	El tipus diccionari associa una clau a un o més elements, per exemple la clau 1 ens donarà 'a', la 2 'b', etc.

1.2. Dades: variables, tipus, constants

```
Python 3.8.8 (default, Apr 13 2021, 19:58:26)
Type "copyright", "credits" or "license" for more information.

IPython 7.22.0 -- An enhanced Interactive Python.

In [1]: x=12

In [2]: type(x)
Out[2]: int

In [3]: x
Out[3]: 12
```

Exemple d'us de tipus i type( )

1.2. Dades: variables, tipus, constants

```
In [37]: alumne=("Joan",17,True)

In [38]: type(alumne)
Out[38]: tuple

In [39]: nom = alumne[0]

In [40]: nom
Out[40]: 'Joan'

In [41]: edat = alumne[1]

In [42]: edat
Out[42]: 17

In [43]: aprovat = alumne[2]

In [44]: aprovat
Out[44]: True
```

Exemple d'us del tipus tupla

1.2. Dades: variables, tipus, constants

```
In [80]: llista = [ 1, 3, [2,4,6], 'a']

In [81]: llista [0]
Out[81]: 1

In [82]: llista [2]
Out[82]: [2, 4, 6]

In [83]: llista [2][2]
Out[83]: 6

In [84]: llista [3]
Out[84]: 'a'
```

Exemple d'us del tipus llista (!!)

1.2. Dades: variables, tipus, constants

Tipus mutables i immutables (!!)

	Tipus mutables	Tipus immutables
Simples		int, long, float, complex, bool
Compostos	list, range, set, dict	string, tuple

Un tipus mutable és aquell que permet, en temps d'execució, canviar-ne el contingut. En canvi, un tipus immutable no pot ser modificat una vegada l'hem creat.

1.2. Dades: variables, tipus, constants

Tipus mutables i immutables (!!)

```
In [24]: x=12

In [25]: type(x)
Out[25]: int

In [26]: id(x)
Out[26]: 93856578502304

In [27]: x=6

In [28]: id(x)
Out[28]: 93856578502112
```

Exemple 1 amb nombre enter

1.2. Dades: variables, tipus, constants

Tipus mutables i immutables (!!)

```
In [30]: llista = [2,4,5,7,8]

In [31]: type(llista)
Out[31]: list

In [32]: id(llista)
Out[32]: 139999930582272

In [33]: llista[0]=1

In [34]: llista
Out[34]: [1, 4, 5, 7, 8]

In [35]: id(llista)
Out[35]: 139999930582272
```

Exemple 1 amb una llista

1.2. Dades: variables, tipus, constants

Tipus mutables i immutables (!!)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

def canvi(x):
    x=65

x=4
canvi(x)

print(x)
```

Exemple 2 amb nombre enter

1.2. Dades: variables, tipus, constants

Tipus mutables i immutables (!!)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

def canvi(l):
    l[2]=65

llista=[1,2,3,5,6]
canvi(llista)

print(llista)
```

Exemple 2 amb una llista

1.2. Dades: variables, tipus, constants

Tipus mutables i immutables (!!)

Similitud amb el pas per valor (immutables) i referència (mutables) en el llenguatge C.

1.2. Dades: variables, tipus, constants

Constants en Python

No hi ha una previsió per definir constants, seran per conveni nostre (nom, majúscules, guions, etc.):

```
RADI_RODA = 12.2
```

El llenguatge Python té predefinides algunes constants com els valors booleans True i False, None, etc.

Si volem emprar constants matemàtiques usarem llibreries com ara numpy.

1.3. Operacions aritmètiques

Assignació

En Python utilitzarem el símbol '=' per assignar (i crear) un valor a una variable.
x = 3 (int), y = 2.12 (float), z = [2,3,4] (llista).

Els operadors aritmètics són els següents:

- Suma '+' : x = x + 3, y = 4 + x
- Resta '-' : y = y - 1, x = 4.5 - 2.0
- Negació '-': y = - x
- Multiplicació '*' : x = x * 3, y = x * 4
- Divisió '/' : x = x / 2, z = z / 4.0
- Divisió entera '//' : x = x // 2 (torna la part entera de la divisió)
- Potència '**' : x = 4 ** 2, y = x ** 3
- Mòdul (reste de la divisió): x = 7 % 2, y = 11 % x

1.3. Operacions aritmètiques

Precedència d'operadors

Potència (**) , Negació (-), Producte (*, /, %, //), Suma (+, -)

Combinació assignació / operació

Exemple: x += 4

Aquesta combinació la podem fer amb tots els operadors aritmètics, per tant tindrem: '+=', '-=', '*=', '/=', '**=', '//=' i '%='

Exemple 2:

x = 7
x **= 2
x +=1 (El valor final de x, serà 50: $x = 7^2 + 1$)

1.3. Operacions aritmètiques

Exercici 1.3.1: Calcula el valor de la variable y després d'executar aquest codi,

```
x = 12
y = x % 5
y **= 4
```

Exercici 1.3.2: Calcula el valor de les variable x i y després d'executar aquest codi,

```
x = 10
y = x ** 2 + 3
y -= 5
x = ( y // x )
```

1.4. Booleans, cadenes, conjunts i llistes

Operacions amb booleans

Les variables de tipus Booleà tenen dos valors possibles: True i False.

Assignació:

b = False
b = (a>3) (b valdrà True si a>3 i False en els altres casos)
c = True

Altres operacions:
c = a and b
c = a or b
a = not b

1.4. Booleans, cadenes, conjunts i llistes

Exercici 1.4.1: Calcula el valor final de la variable b després d'executar aquest codi,

```
x = 12
        a = (x > 10) and (x < 20)
        b = not a
```

1.4. Booleans, cadenes, conjunts i llistes

Operacions amb cadenes

Assignar:

```
s = “Aprenc Python”
```

Concatenar (+) dues variables de tipus string:

```
s1 = “Hola“
s2 = “amics”
s3 = s1 + s2    (s3 valdrà “Holaamics”)
```

Repetir (*) :

```
s1 = “Hola“
s2 = s1 * 3    (s2 valdrà “HolaHolaHola”)
```

1.4. Booleans, cadenes, conjunts i llistes

Operacions amb cadenes

Podem tallar ([ ]) trossos d'un string:

s1 = "Hola amics"
s2 = s1[3] (s2 valdrà "a")
s2 = s1[3:7] (s2 valdrà "a am")
s2 = s1[-2] (s2 valdrà "c")

Cadena	H	o	l	a		a	m	i	c	s
Posició +	0	1	2	3	4	5	6	7	8	9
Posició -	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

1.4. Booleans, cadenes, conjunts i llistes

Operacions amb cadenes

Pertanyença (in / not in) dins d'un string:

```
s1 = "Python és genial"
b = "Python" in s1 (b valdrà True)
b = "Java" in s1    (b valdrà False)
```

Comparar (== / !=) dos strings per saber si són iguals:

```
s1 = "Python és genial"
b = ("Python és fantàstic" == s1)      (b valdrà False)
```

1.4. Booleans, cadenes, conjunts i llistes

Exercici 1.4.2.1: Donada la cadena “Hola bon dia, com esteu?”, trobeu la x i la y per extreure el segment “bon dia”

s1 = “Hola bon dia, com esteu?”
s = s1 [x : y]

Exercici 1.4.2.2: Quin resultat tindrem a la variable s si fem servir aquest codi?

s1 = “Hola bon dia, com esteu?”
s = s1 [4: -5]

1.4. Booleans, cadenes, conjunts i llistes

Funcions aplicables al tipus cadena

s= ”Python és genial”

- len() x=len(s) torna la longitud de la cadena (en aquest cas 16).
- s.capitalize() convertirà el primer caràcter a lletres majúscules.
- s.upper() passa tota la cadena a lletres majúscules.
- s.lower() passa tota la cadena a lletres minúscules.
- s.count(cadena) retorna quants cops hi ha cadena dins de s.
- s.find(cadena) cerca cadena dins de s i ens torna la posició on és.
- s.isalpha() ens retorna True si tots els caràcters són de l’alfabet.
- s.isdigit() ens retorna True si tots els caràcters són nombres 0-9.
- s.isalnum() retorna True si tots els caràcters són lletres o nombres.
- s.split(separador) torna una llista amb s partit en trossos segons separador.

A l’exemple si cridem s.split(“ “) torna una llista amb els elements [“Python”,”és”,”genial”]

1.4. Booleans, cadenes, conjunts i llistes

Operacions amb conjunts

El tipus conjunt permet tenir elements no repetits i sense un ordre concret.

```
colors1 = { “blau”, ”verd”, ”vermell”, ”groc”}
colors2 = { “verd”, “negre”, “blanc”}
```

- unió (|) permetrà unir dos conjunts:
  colors3 = colors1 | colors2 (colors3 tindrà 6 elements)

- intersecció (&) calcularà els elements dels dos conjunts:
  colors3 = colors1 & colors2 (colors3 tindrà 1 element, “verd”)

- diferència (-) eliminarà del primer conjunt els elements del segon:
  colors3 = colors1 - colors2
  (colors3 tindrà 3 elements, “blau”, ”vermell” i “groc”)

1.4. Booleans, cadenes, conjunts i llistes

Funcions aplicables al tipus conjunt

c1 = { “blau”, ”verd”, ”vermell”, ”groc”}
c2 = { “blau”, ”verd”, ”vermell”}

-	len()	x=len(c1) ens tornarà el nombre d’elements d’un conjunt (4)
-	c1.add(elem)	Afegirà un element nou (elem) al conjunt c1.
-	c1.remove(elem)	Eliminarà l’element (elem) del conjunt c1.
-	c1.copy()	Ens retornarà una còpia del conjunt, exemple: d = c1.copy( )
-	c1.clear()	Elimina tots els elements del conjunt c1.
-	c1.intersection(c2)	Retorna la intersecció entre els conjunts c1 i c2. En aquest cas: “blau”, “verd”, “vermell”.
-	c1.union(c2)	Retorna la unió entre els conjunts c1 i c2. En aquest cas: “blau”, “verd”, “vermell”, “groc”.
-	c1.difference(c2)	Retorna la diferència entre els conjunts c1 i c2: “groc”

1.4. Booleans, cadenes, conjunts i llistes

Exercici 1.4.3.1: Donats els següents conjunts:

s1 = { ‘A’, ‘B’, ‘D’}
s2 = { ‘B’, ‘C’, ‘D’}
s3 = { ‘A’, ‘E’, ‘I’, ‘O’, ‘U’}

Quin resultat tindrem al conjunt s després de fer:

s = (s1 | s3) - s2

1.4. Booleans, cadenes, conjunts i llistes

Operacions amb llistes **Operacions mutables**

- Afegir (append) un element a una llista. L'afegirà pel final.

```
x = [ 1, 2, 3 ]
x.append (5)                    ( x tindrà ara [ 1, 2, 3, 5 ])
```

- Estendre (extend) afegeix una llista a una llista. L'afegirà pel final.

```
x = [ 1, 2, 3 ]
x.extend ( [4,5] )              ( x tindrà ara [ 1, 2, 3, 4, 5 ])
```

- Inserir (insert) afegeix un element a una posició concreta d'una llista.

```
x = [ 1, 2, 3, 4, 5 ]
x.insert ( 2, 12 )              ( x tindrà ara [ 1, 2, 12, 3, 4, 5 ])
```

- Eliminar (del) elimina un element d'una posició concreta d'una llista.

```
x = [ 1, 2, 3, 4, 5 ]
x.del ( 2 )                     ( x tindrà ara [ 1, 2, 4, 5 ])
```

1.4. Booleans, cadenes, conjunts i llistes

Operacions amb llistes **Operacions mutables (2)**

- Eliminar (remove) elimina la primera aparició d'un element d'una llista.
  x = [1, 2, 3, 2, 1]
  x.remove (2) (x tindrà ara [1, 3, 2, 1])

- Girar (reverse) dóna la volta a una llista, el primer serà l'últim, etc.
  x = [1, 2, 3, 4, 5]
  x.reverse ( ) (x tindrà ara [5, 4, 3, 2, 1])

- Ordenar (sort) Ordena una llista de petit a gran (és la opció per defecte).
  x = [1, 4, 3, 5, 2]
  x.sort ( ) (x tindrà ara [1, 2, 3, 4, 5])

1.4. Booleans, cadenes, conjunts i llistes

Operacions amb llistes **Operacions mutables (3)**

- Ordenar (sort) Ordena una llista de petit a gran (és la opció per defecte).

La funció sort( ) pot tenir un paràmetre sort (reverse= True) per ordenar de gran a petit. Els elements de la llista han de ser del mateix tipus, si volem ordenar una llista de cadenes ho farà per ordre alfabètic.

La funció sort, permet passar-li com a paràmetre una funció (key), per definir un criteri d'ordenació diferent (per exemple, llargada d'una cadena). Podem usar una funció preexistent o definir una pròpia (això es veurà més endavant).

```
x = [ 'Anna', 'Xi', 'Fatima' ]
x. sort (key = len)
```

(x valdrà ara ['Xi', 'Anna', 'Fatima'])

1.4. Booleans, cadenes, conjunts i llistes

Operacions amb llistes

Exercici 1.4.4.1: Donades les següents llistes:

n1 = [‘Juan’, ‘Enric’, ‘Sarai’]
n2 = [‘Ahmed’, ‘Laura’]
n3 = [‘Liu’, ‘Antonio’, ‘Maria’]

Generar una única llista noms[ ] amb els noms de n1, n2 i n3 ordenats alfabèticament.

1.4. Booleans, cadenes, conjunts i llistes

Operacions amb llistes **Operacions immutables**

- Ordenar (sorted) Retorna una nova llista ordenada de petit a gran.

  x = [1, 4, 3, 5, 2]
  y = sorted (x) (y tindrà ara [1, 2, 3, 4, 5] mentre que x seguirà tenint els valors [1, 4, 3, 5, 2])

- Afegir (+) permet afegir una llista a una altra.

  x = [1, 2, 3, 4]
  y = [5, 6]
  z = x + y (z tindrà ara [1, 2, 3, 4, 5, 6])

- Replicar (* n) Replica els elements d'una llista n vegades.

  x = [1, 2, 3]
  y = x * 3 (x tindrà ara [1, 2, 3, 1, 2, 3, 1, 2, 3])

1.4. Booleans, cadenes, conjunts i llistes

Operacions amb llistes **Operacions immutables (2)**

- Mínim (min) i màxim (max) retorna l'element més petit o més gran d'una llista.

```
x = [ 3, 2, 3, 4, 7, 4 ]
a = min ( x )          ( a serà un enter, i valdrà 2 )
b = max ( x )          ( b serà un enter, i valdrà 7 )
```

- Localitzar (index) retorna la posició d'un element en una llista.

```
x = [ 3, 6, 2, 5, 1, 7, 4, 8 ]
a = x.index (5)        ( a valdrà 3, és a la posició 3 començant per 0)
```

- Comptar (count) torna quants cops un element és dins d'una llista.

```
x = [ 1, 2, 5, 4, 5, 1, 3, 5, 7, 7 ]
a = x.count (5)              ( a valdrà 3 perquè 5 apareix tres cops )
                             ( retornarà 0 si no hi és)
```

1.4. Booleans, cadenes, conjunts i llistes

Operacions amb llistes **Operacions immutables (3)**

- Existència (in) informa amb un booleà si un element és dins d'una llista.

  x = [1, 2, 5, 4, 5, 1, 3, 5, 7, 7]
  a = 8 in x
  b = 4 in x (a valdrà False, b valdrà True)

- Suma (sum) suma els elements d'una llista si són numèrics.

  x = [1, 2, 5, 4, 5, 1, 3, 5, 7, 7]
  a = sum (x) (a valdrà 40)

- Llargària (len) compta el número d'elements d'una llista.

  x = [1, 2, 5, 4, 5, 1, 3, 5, 7, 7]
  a = len (x) (a valdrà 10)

1.5. Interacció amb l'usuari

La funció print ( ) Pintarà el que li donguem.

```
x=12

llista=[1,2,3,4,5]

s="Patata"

LN=["Moha","Ke-hon","Pere","Jana","Alba","Ian"]

print ("Proves:")
print (3)
print (x)
print (llista)
print (s)
print (LN)
```

```
3
12
[1, 2, 3, 4, 5]
Patata
['Moha', 'Ke-hon', 'Pere', 'Jana', 'Alba', 'Ian']
```

1.5. Interacció amb l'usuari

La funció print ( ) (exemples, decoració i separadors)

```
print ("La llista és:",llista)
print ("La paraula és:",s)
print ("El nombre és:",x)


a=2
b=4
c=6

print ( a, b, c)
print ( a, b, c, sep=';')
```

La llista és: [1, 2, 3, 4, 5]
La paraula és: Patata
El nombre és: 12

Obtenim: 2 4 6
2;4;6

1.5. Interacció amb l'usuari

La funció print ( ) (exemples, canvi de finalització)

```
a=2
b=4
c=6

s="Patata"


print (a,b,c)
print (s)

print (a,b,c,end='---')
print (s)
```

Obtenim:

```
2 4 6
Patata
2 4 6---Patata
```

Enlloc d'acabar amb salt ('\n') podem canviar-ho per un altre caràcter o seqüència.

Exercici 1.5.1.1: Escriure amb un sol print la frase "Python és divertit !!" deixant dues línies en blanc davant i dues darrere.

1.5. Interacció amb l'usuari

La funció input ( ) Llegirà l'entrada de teclat

```
n = input ("Entra un nombre: ")
print ("Has entrat:",n)
```

A l'utilitzar la funció input cal tenir en compte les següents consideracions:

- la funció és **bloquejant**. Fins que l'usuari no entri un valor i premi la tecla 'Enter' el programa quedarà en aquesta línia de codi.

- sempre torna una variable de tipus **cadena** (string).
  A l'exemple, no ha tornat el nombre 89 sinó que n tindrà una cadena amb dos caràcters: 8 i 9.

1.5. Interacció amb l'usuari

La funció input ( ) Llegirà l'entrada de teclat

```
n = input ("Entra un nombre: ")
print (type(n))
x=int(n)
print (type(x))
print ("Has entrat:",x)
```

Podrem fer una conversió de tipus, com a l'exemple. Normalment això s'anomena ***casting***.

1.5. Interacció amb l'usuari

Aquests són algunes de les **conversions de tipus** que podem fer en Python:

- a = int (s) Convertirà s (cadena) a nombre enter.
  Si s no correspon a un enter, per exemple s="hola", tornarà un error.

- a = int (f) Convertirà f (float, nombre real) a enter. Atenció: ho fa truncant!! per tant si f=23.99, a valdrà 23.

- b = float (s) Convertirà s (cadena) al nombre real més proper.
  Si s no correspon a un nombre, exemple s="hola", tornarà un error.

- b = float (a) Si a és un enter, b serà el real equivalent (ex. a = 3 -> b = 3.0)

- c = str (a) Convertirà el nombre a, a la cadena de caràcters que el representa.

1.5. Interacció amb l'usuari

Exercici 1.5.2.1: Demanar un nombre enter per pantalla i pintar el nombre i el seu doble.

Exercici 1.5.2.2: Demanar dos nombres enters per pantalla i pintar els nombres i la seva suma.

1.6. Fitxers, guardar dades

Fitxers en Python

El llenguatge Python disposa de comandes per crear, obrir, escriure i llegir fitxers.

Com en tots els llenguatges de programació, aquestes comandes reben suport del **sistema operatiu** que tinguem a sota (Linux, Windows, macOS, Android, etc.) per tant els llocs on els nostres fitxers aniran a parar, estaran sota l'estructura del sistema operatiu.

1.6. Fitxers, guardar dades

La funció open ( )

f = open (nom_fitxer, mode)

La funció open ens tornarà una variable especial (aquí anomenada f, però pot tenir el nom que vulguem), que ens permetrà utilitzar el fitxer. El nom de fitxer serà una cadena on especificarem el nom que volem donar al fitxer.

Exemples:

f = open (“ Prova.txt “)
Obrirà el fitxer anomenat Prova.txt a la carpeta actual.

f = open (“ C:\usuari\dades\Prova.txt”)
Obrirà el fitxer Prova.txt a la carpeta, estil Windows.

f = open (“ ./data/Prova.txt”)
Obrirà el fitxer Prova.txt a la carpeta, estil Linux.

1.6. Fitxers, guardar dades

La funció open ( )

A més del nom, indicarem a la funció open un mode de funcionament:

- r lectura (reading), és el mode per defecte.
- w escriptura (writing). Si no existeix el fitxer, el crea. Si ja existeix el sobreescriurà.
- a afegir (append). Si no existeix el crea. Si existeix manté el contingut.
- + actualitzar (update). Obre el fitxer per lectura i escriptura.

- t texte (text). El fitxer estarà en mode text, és a dir amb caràcters alfanumèrics.
- b binari (binary). El fitxer estarà en mode binari, 1 i 0 directament sense cap codificació especial.
- x exclusiu (exclusive). Obrirem el fitxer de manera exclusiva. Mentre hi estiguem treballant, cap altre programa del sistema operatiu el podrà utilitzar.

1.6. Fitxers, guardar dades

La funció close ( )

Quan acabem acabat de treballar amb el fitxer: llegir-lo, escriure'l o afegir-hi contingut, el tancarem per informar al sistema operatiu que està disponibile per altres programes. Sempre cal tancar un fitxer que hem obert, sinó les dades es poden perdre. Per tancar el fitxer és tan senzill com cridar la funció close ( ) associada al manegador de fitxers que ens ha tornat open ( ).

Exemple:

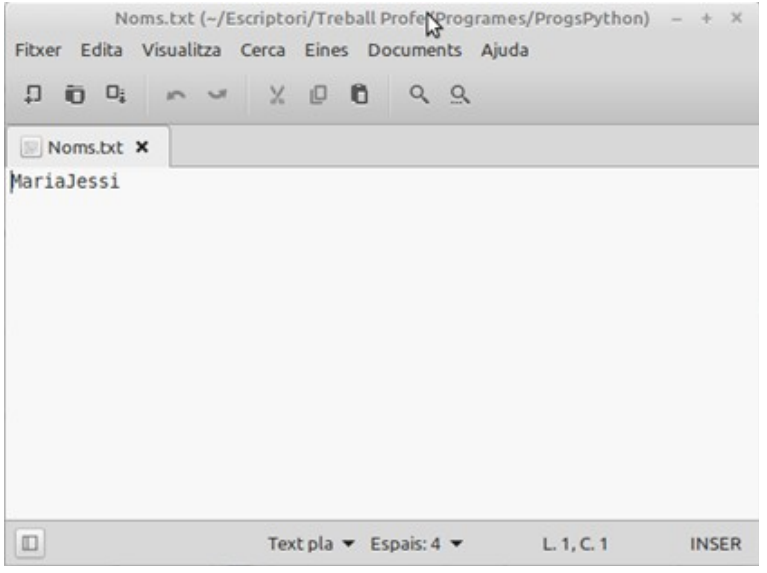
```
f = open ( " Dades_temperatura.txt", "wt")
# operacions de recollir i escriure dades al fitxer
f. close ( )
```

1.6. Fitxers, guardar dades

La funció write ( )

Aquesta és la funció que ens permetrà escriure dades al nostre fitxer. Com a la funció close, write està associat al manegador de fitxers que ens ha tornat open. Li passarem un paràmetre que seran les dades que aniran al fitxer. Es pot cridar diverses vegades.

```
f = open ( " Noms.txt", "wt")
f.write ( "Maria" )
f.write ( "Jessi" )
f. close ( )
```

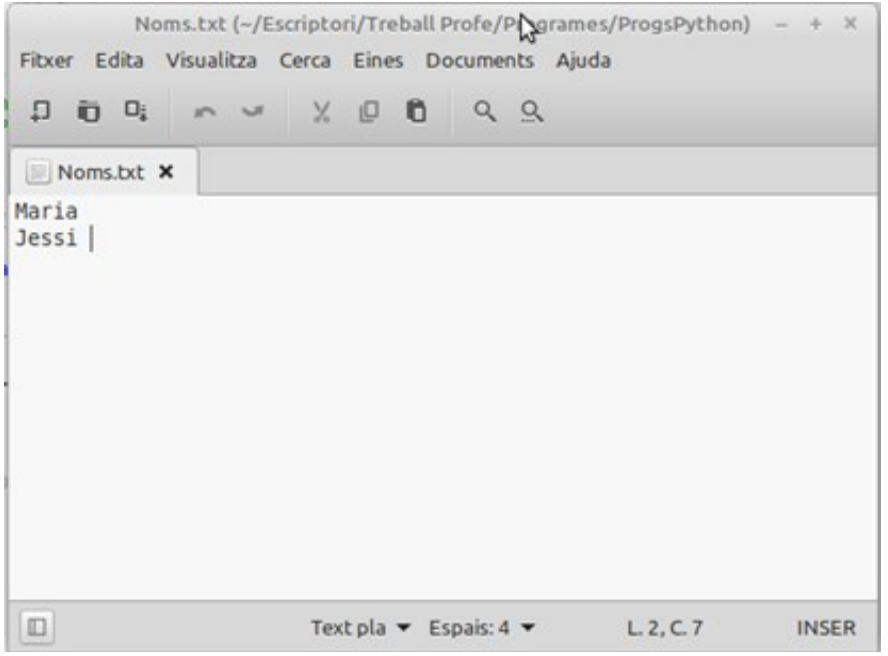


1.6. Fitxers, guardar dades

La funció write ( )

Podem controlar el format!!

```
f = open ( " Noms.txt", "wt")
f.write ( "Maria \n" )
f.write ( "Jessi \n" )
f. close ( )
```



1.6. Fitxers, guardar dades

Les funcions readline ( ) i read( )

La funció readline ( ) llegeix directament una línia del fitxer i ens la torna com a cadena de caràcters (string), per tant és ideal com a complement del codi que acabem de veure. Si tenim un fitxer amb dos noms de persones com l'anterior, un programa com aquest:

```
f = open ( "Noms.txt", "rt")
s1 = f.readline ( )
s2 = f.readline ( )
print ( "El primer nom és: ",s1)
print ( "El segon nom és: ",s2)
f. close ( )
```

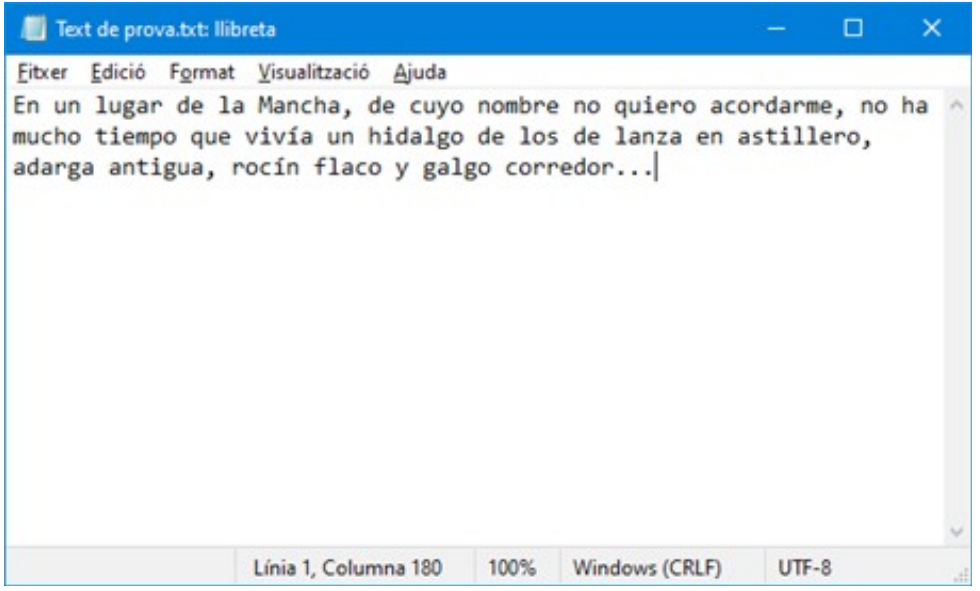
Mostrarà directament a pantalla:

El primer nom és: Maria
El segon nom és: Jessi

1.6. Fitxers, guardar dades

Les funcions readline ( ) i read( )

La funció read (n) permet llegir n bytes d'un fitxer. Per fitxers de texte, podem assumir que cada caràcter ocupa un byte. Si tenim un fitxer com aquest:



1.6. Fitxers, guardar dades

Les funcions readline ( ) i read( )

I executem:

```
f = open ( "Text de prova.txt", "rt")
s = f.read (24)
print ( s )
f. close ( )
```

Tindrem a la sortida:

En un lugar de la Mancha

Complementarem read ( ) amb:

La funció: f.tell ( ) que donat un fitxer f ens diu per quin caràcter anem.
La funció: f.seek (n) que ens col·loca el punt de lectura del fitxer a la posició n.

1.7. Funcions

Pròposit: Organització i simplificació del codi.

Us de ‘def’ i ‘return’.

Exemple:

```
def suma (a , b):
    r = a + b
    return (r)

c = suma (12,3)

print (c)
```

1.7. Funcions

Pròposit: Organització i simplificació del codi.

Exemple:

```
def suma (a , b):
    r = a + b
    return (r)


a=7
a = suma (a,12)
a = suma (a,25)
a = suma (a,6)

print (a)
```

1.7. Funcions

Pròposit: Organització i simplificació del codi.

Permet retornar múltiples valors:

```
def opera (a , b):
    s = a + b
    r = a - b
    p = a * b
    q = a / b
    return (s,r,p,q)


a,b,c,d = opera (18,6)

print (a,b,c,d)
```

1.7. Funcions

Pròposit: Organització i simplificació del codi.

Les seves variables són locals.

```
def suma (a , b):
    r = a + b
    return (r)

a=7
b=12
r=13
c = suma (a,b)

print (c)
print (r)
```

1.7. Funcions

Pròposit: Organització i simplificació del codi.

Compte amb tocar els tipus mutables:

```
def valormig (a):
    n= len(a)
    s= sum(a)
    return (s/n)


l=[1,3,2,5,6,5]

print (valormig(l))
print (l)
```

1.8. Bifurcacions, repeticions

Bifurcacions

Els llenguatges de programació ofereixen una estructura per poder fer bifurcacions en l'execució del codi.

- S'avaluarà una condició (o una combinació de condicions) (and, or…)
- Es proposarà una o més instruccions a executar si la condició és certa
- Es proposarà (opcionalment) una o més instruccions a executar si la condició és falsa.

1.8. Bifurcacions, repeticions

Bifurcacions. If, else, elif. Exemples

```
if a%2==0:
    print("És parell")
else:
    print("És senar")
```

```
if a<0:
    print("És negatiu")
elif a>0:
    print("És positiu")
else:
    print("És zero")
```

1.8. Bifurcacions, repeticions

Bifurcacions. Exemples

```
if a==0:
    print("És zero")
elif a==1:
    print("És u")
elif a==2:
    print("És dos")
elif a==3:
    print("És tres")
else:
    print("És quatre")
```

Atenció amb les extensions d'if !! (temps de còmput, organització...)

```
dic={0:"Zero",1:"U",2:"Dos",3:"Tres",4:"Quatre",5:"Cinc",
     6:"Sis",7:"Set",8:"Vuit",9:"Nou"}

print ("És " + dic[a])
```

1.8. Bifurcacions, repeticions

Repeticions: While

L'estructura de programació amb while ens permetrà repetir un bloc de codi mentre es dongui una condició.

```
while condició :
        instruccions
```

Com en el cas de l'if, pot haver una o diverses instruccions dins del **bloc** del 'while' que vindran agrupades també amb un **tabulador** (tot el bloc ha d'estar tabulat).
Cal anar amb compte: **si la condició sempre és certa, crearem un bucle infinit** per tant caldrà pensar bé les condicions que s'escriuen.

1.8. Bifurcacions, repeticions

Repeticions: While Exemples

```
i=0
while i<10:
    print(i)
    i=i+1
```

```
dic={0:"Zero",1:"U",2:"Dos",3:"Tres",4:"Quatre",5:"Cinc",
     6:"Sis",7:"Set",8:"Vuit",9:"Nou"}

i=0
while i<10:
    print (dic[i])
    i=i+1
```

1.8. Bifurcacions, repeticions

Repeticions: While Mal exemple

```
dic={0:"Zero",1:"U",2:"Dos",3:"Tres",4:"Quatre",5:"Cinc",
     6:"Sis",7:"Set",8:"Vuit",9:"Nou"}

i=0
while i<10:
    print(i , "," , dic[i])
```

Per què?

1.8. Bifurcacions, repeticions

Repeticions: While **Mals hàbits!**

La comanda continue salta el cas actual d'un bucle:

```
i=0
while i<10:
    i=i+1
    if i==6:
        continue
    print(i)
```

La comanda break permet sortir d'un bucle en qualsevol moment:

```
i=0
while i<10:
    if i==6:
        break
    print(i)
    i=i+1
```

1.8. Bifurcacions, repeticions

Repeticions: for

La sentència for, permetrà recórrer TOTS els elements d'una estructura iterable: una llista, una tupla, un conjunt, un diccionari, etc.

La seva utilització és:

```
for element in tipus_iterable:
    instruccions
```

1.8. Bifurcacions, repeticions

Repeticions: for

```
llista=["Primavera","Estiu","Tardor","Hivern"]

for el in llista:
    print(el)
```

Exemple de recorregut i recorregut amb operació:

```
L1=["Juan","Maria","Pere","Ian","Antònia"]
L2=["Moha","Ke-hon","Pere","Jana","Alba","Ian"]

for el in L1:
    if el in L2:
        print(el)
```

1.8. Bifurcacions, repeticions

Discussió: tria **while** o **for**

```
llista=[]

i=1
while i<=10:
    llista.append(i**2)
    i=i+1

print(llista)
```

vs

```
llista=[]

for i in range(1,11):
    llista.append(i**2)

print(llista)
```

Resultat idèntic : [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

1.8. Bifurcacions, repeticions

Exercici 1.8.3.2: Agafar tots els nombres entre 0 i 1000 i ordenar-los en quatre llistes:

- **múltiples de 2**
- **múltiples de 3**
- **múltiples de 5**
- **resta de nombres**

Pintar aquestes 4 llistes per pantalla.

1.8. Bifurcacions, repeticions

Exercici 1.8.3.3: Aquest exercici consta de dues fases:

- **Fer un programa que demani noms de persones i els vagi afegint a una llista inicialment buida fins que el nom sigui “sortir”.**
- **Escriure tots els noms de persones amb l’afegit “és a la llista”.**

Exemple:

Llista = [“Javi”,”Joana”]
Escriure:
Javi és a la llista.
Joana és a la llista.

1.9. Comentaris

Comentari de bloc o de línia:

```
'''

Aquest codi crearà dues variables num1 i num2
de tipus enter i en calcularà la suma, que es
guardarà a res

'''

num1 = 23
num2 = 36

res = num1 + num2 # a res queda el resultat.
```

Utilitzar per descriure i aclarir conceptes, per nosaltres mateixos en el futur o altres persones.