

Algorísmica

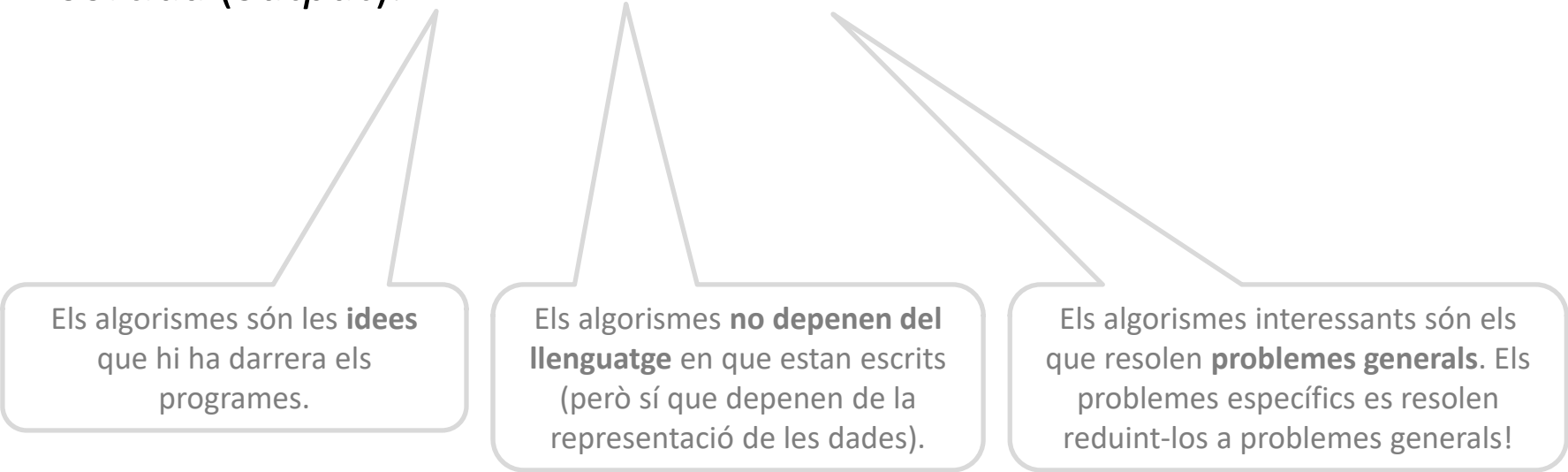
Què és un algorisme?

Mireia Ribera & Jordi Vitrià

Què és algorisme?

Un **algorisme** és **una seqüència finita, no ambigua i explícita, d'instruccions per a resoldre un problema.** (*Wikipedia*)

La definició d'aquesta assignatura: Un algorisme és qualsevol procediment computacional que pren un (o una sèrie) de valors com a entrada (*input*) i genera algun valor (o conjunt de valors) com a sortida (*output*).



Els algorismes són les **idees** que hi ha darrera els programes.

Els algorismes **no depenen del llenguatge** en que estan escrits (però sí que depenen de la representació de les dades).

Els algorismes interessants són els que resolen **problemes generals**. Els problemes específics es resolen reduint-los a problemes generals!

Exemples “no computacionals”

HOW TO EAT A BELLY BUSTER BURRITO™

The 4-Point Approach

Peeling Process

1. In a circular motion, remove foil an inch or two at a time. Expose just enough of your Belly Buster Burrito™ to eat. Eat. Repeat as necessary.

2 Attack Angle

Hold burrito upright. Begin eating at one end. Continue eating until reaching the other end.

3 Support Structure

Remaining foil keeps burrito warm, intact, and out of your lap.

4 Practice Schedule

As with other advanced skills, there's no substitute for experience. You'll enjoy increased performance with each visit to Belly Buster Burritos™

WARNING: Belly Buster Burritos™ are hot. To fully enjoy your eating experience, please be careful if driving or operating heavy machinery larger than one of our forks. After all, it'd be a shame to spill a meal this good.

BUILDING A HUMANURE TOILET

1. start with four identical buckets
standard toilet seat
hole fits bucket rim
2 hinges
3/4" plywood 18"x18"
3/4"x18"x3" board
(2) 3/4"x10"x18"
(2) 3/4"x10"x19.5"
(4) 3/4"x3"x12"
2. screw boards together
box is 10" deep, 18" wide and 21" long
3. Screw 3"x18" board to box. Leave 18"x18" plywood loose on hinges
4. Screw legs to inside of box. Bucket MUST protrude through plywood by 1/2". Adjust legs accordingly.
5. Swivel plastic bumpers sideways so top of bucket rim will fit against toilet seat.
6. adjusted toilet seat

Exemple computacional (arrel quadrada)

Input:

Un nombre a

Output:

Un nombre b tal que $b*b=a$

Observació:

Volem una solució **correcte** i **eficient**!

Exemple computacional (arrel quadrada)

Heró d'Alexandria (10 dC-70 dC) va proposar el següent algorisme:

1. Comencem amb un nombre qualsevol g .
2. Si $g*g$ s'assembla prou a a , ens aturem i donem la resposta.
3. Sinó, calculem un nou candidat $(g+a/g)/2$.
4. Anem repetint aquest procés fins que ens aturem.



```
def heron(n, error):  
    import math  
    g = 1.0  
    while math.fabs(g*g - n) > error:  
        g = 1/2*(g+n/g)  
        print(g)  
    return
```

```
heron(49,0.001)
```

```
25.0  
13.48  
8.557507418397627  
7.141736912383411  
7.001406475243939  
7.000000141269659
```

Correcció i Eficiència Algorísmica

Un algorisme es **correcte** si podem **demostrar** que retorna la sortida desitjada per a qualsevol entrada legal (per al problema de l'arrel quadrada, això vol dir nombres positius o 0!).

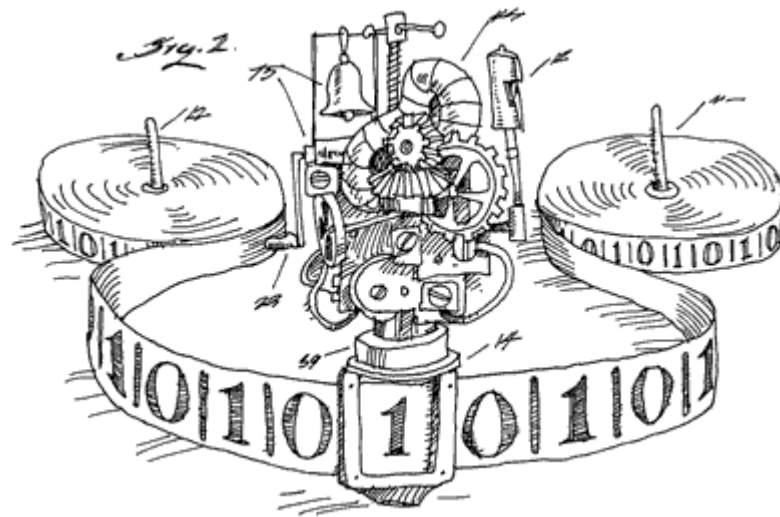
És **eficient** si es fa amb el **mínim nombre de recursos** (**cicles de càlcul = temps, memòria=espai**) possible.

Demostrar la “correcció” és fàcil per alguns algorismes, difícil per la majoria i fins i tot impossible per alguns.

Algorismes i ordinadors

Un ordinador fa només dues coses (però molt ben fetes!): **calcular** (fer operacions entre dades) i **emmagatzemar** (llegir/escriure) els resultats del càlcul.

Un ordinador convencional fa més de 1.000.000.000 de càlculs per segon i pot emmagatzemar més de 1.000.000.000.000 de bits.



El problema del viatjant de comerç

Aquest cartell correspon al concurs promogut per Procter & Gamble l'any 1962 per recórrer 33 ciutats dels EEUU.

HELP! WE'RE LOST!

HELP "CAR 54" ...AND WIN CASH
54...\$1,000 PRIZES
ONE...\$10,000 GRAND PRIZE

START and FINISH

Map by Rand McNally

Help Toody and Muldoon find the shortest round trip route to visit all 33 locations shown on the map.

All you do is draw connecting straight lines from location to location to show the shortest round trip route.

HERE'S THE CORRECT START...

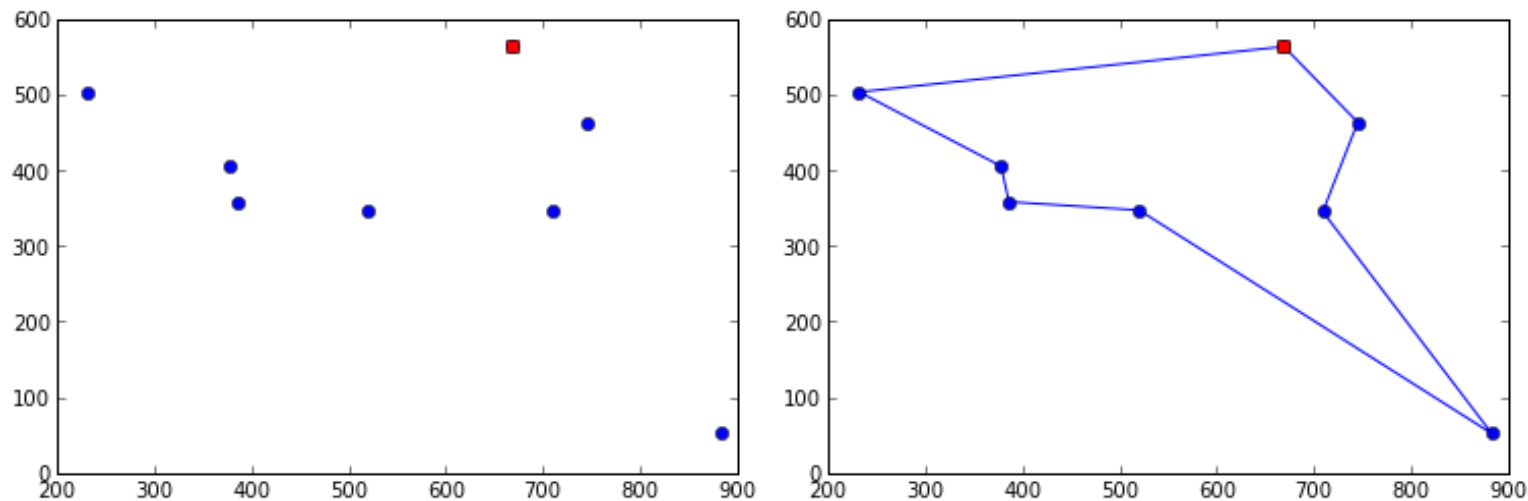
Begin at Chicago, Illinois. From there, lines show correct route as far as Erie, Pennsylvania. Next, do you go to Carlisle, Pennsylvania or Wana, West Virginia? Check the easy instructions on back of this entry blank for details.

© PROCTER & GAMBLE 1962

OFFICIAL RULES ON REVERSE SIDE

Estratègies possibles

Suposem que hem de passar per cada un d'aquests punts i volem minimitzar la distància recorreguda.



Estratègies possibles

Escollir la ciutat inicial és una decisió arbitrària que no afecta al resultat.

Solució I: Escollim un punt aleatori, i anem seleccionant el veí més proper per continuar.

$p = p_0$

$i = 0$

Mentre hi hagi punts per visitar

$i = i + 1$

Indentació

Determinem p_i , el punt més proper a p_{i-1}

Visitem p_i

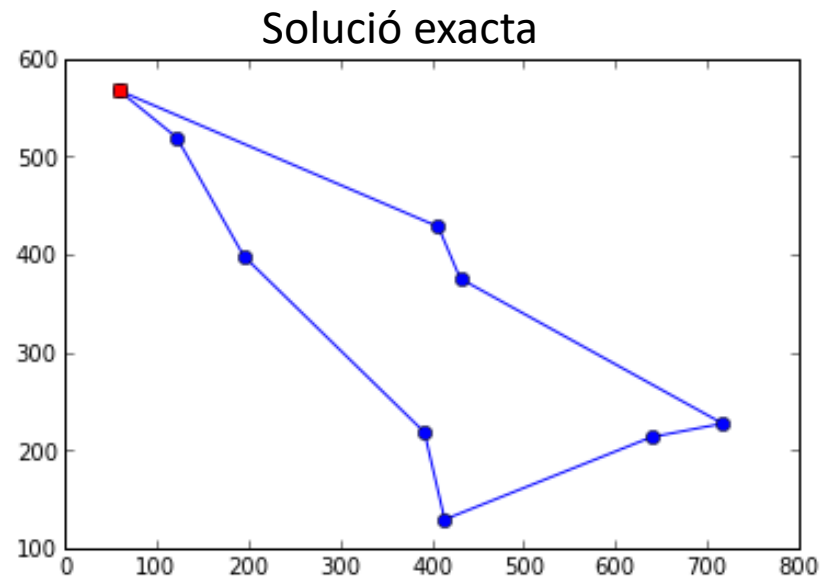
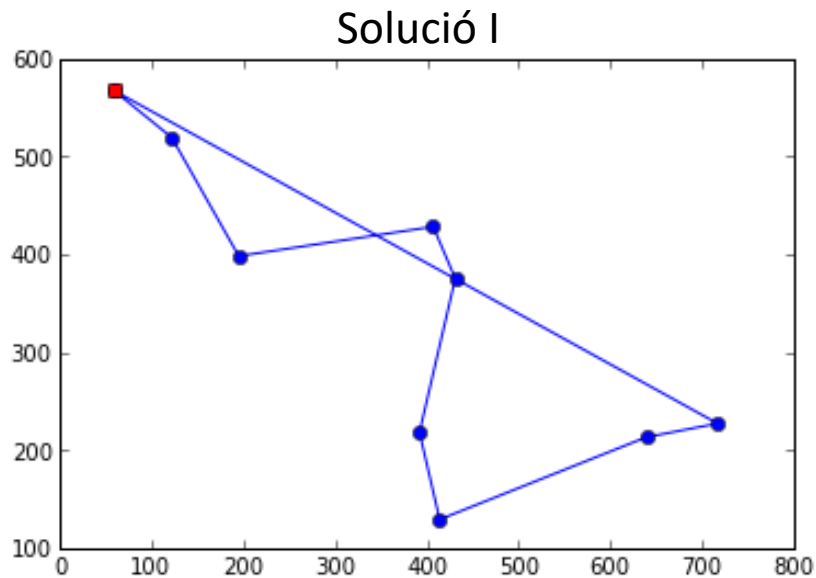
Retorna la seqüència

Algorisme especificat en
pseudocodi

Estratègies possibles

És evident que no és correcte!

No sempre dona el resultat correcte, tot i que per alguns casos particular ho pot fer.



Estratègies possibles

Solució II: Considerem tots els possibles passos parcials entre dues ciutats i anem afegint repetidament el més petit sempre i quan no generi un cicle o una doble sortida per un punt.

$d = \infty$

Per $i=1$ fins a $N-1$

Per cada parella de punts (x,y) no connectats

Si $dist(x,y) \leq d$ llavors

$x_m = x, y_m = y,$

$d = dist(x,y)$

Connecta x_m i y_m amb un camí

Retorna la seqüència

Tampoc és correcte!

Estratègies possibles

Solució III: Considerem totes les possibles ordenacions dels punts i seleccionem la més curta.

$$d = \infty$$

Per cada una de les $n!$ permutacions P_i dels n punts

Si $(cost(P_i) \leq d)$ llavors $d = cost(P_i)$ i $P_{min} = P_i$

Retorna P_{min}

És correcte!, però és eficient?

10! = 3,628,800

```
>>> factorial2.main()
Please enter a whole number: 100
The factorial of 100 is 933262154439441526816992388562667004907159682
643816214685929638952175999932299156089414639761565182862536979208272
237582511852109168640000000000000000000000
```

Com expressem els algorismes?

Amb llenguatges de programació.

Un llenguatge de programació es defineix per unes **primitives** (símbols), una **sintaxi** (regles de combinació de símbols), una **semàntica estàtica** (combinacions de símbols amb significat) i una **semàntica** (el significat que nosaltres volem donar a l'algorisme).

Com expressem els algorismes?

Fins ara hem usat el pseudocodi, però també podem usar un llenguatge d'alt nivell, **Python**, molt proper al pseudocodi, que ens permetrà executar els algorismes!

El preu que hem de pagar és que haurem **d'especificar** una mica més les coses.

Els avantatges: aprenem un llenguatge útil, som més formals en les especificacions, podem fer simulacions.

Llenguatges

- **Sintaxi:** “3.2 + 4.5” vs “3.2 a 2.3”
- **Semàntica estàtica:** 3.2/’abc’ és sintàcticament correcte perquè l’expressió (<literal><operador><literal>) ho és, però no ho és des del punt de vista de la semàntica estàtica.

Els errors més perillosos quan programem no són els sintàctics, atès que la majoria es poden detectar automàticament o són fàcils de veure!

Alguns llenguatges detecten casi tots els errors de semàntica estàtica, però Python només alguns!

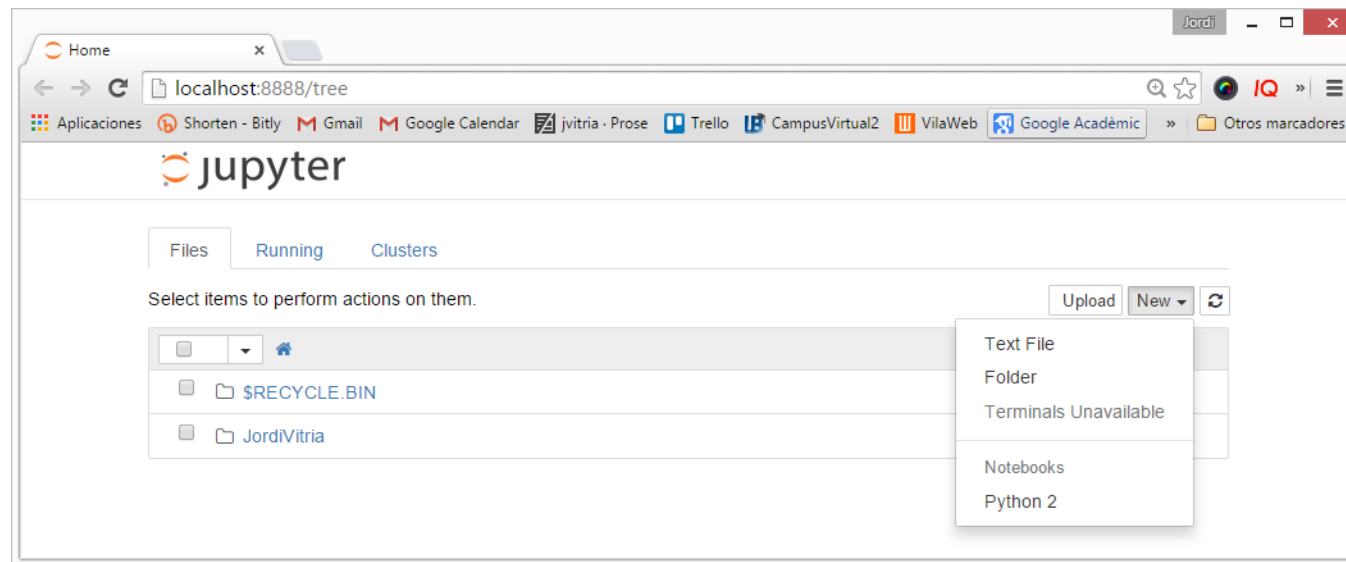
Si no hi ha **errors semàntics el programa farà alguna cosa (no necessàriament la que volem!)**

Llenguatges

Si un programa té un error:

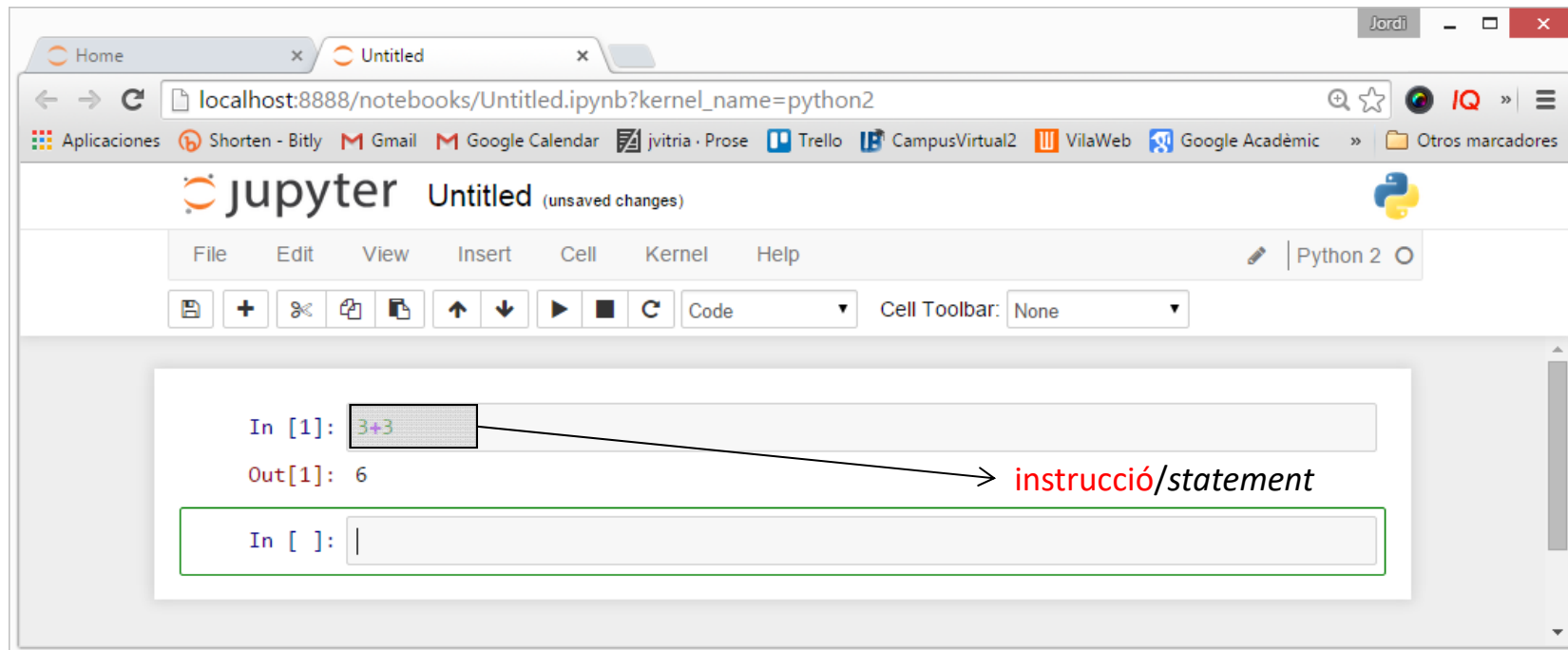
1. Pot acabar bruscament la seva execució i generar un error. La majoria de vegades no afecta a la resta de programes de l'ordinador, però...
2. Pot ser que mai s'aturi.
3. Pot executar-se i generar una resposta que pot ser, o no, correcte.

Suposem que hem instal·lat el llenguatge Python i l'entorn Jupyter Notebook al nostre ordinador:



Un programa en Python

Ja podem començar a donar ordres...



Un programa en Python

Si volem executar una seqüència d'instruccions podem crear/**definir** una **funció** (que en aquest cas s'anomena `hola`):

```
def hola():  
    print("hola")
```

Indentació per indicar
que són part de la
definició de la funció

Un cop la tenim definida la poden **cridar/invocar**:

```
def hola():  
    print("hola")
```

```
hola()
```

```
hola
```

Un programa en Python

Les funcions poden tenir **paràmetres** (que van entre els parèntesi):

```
def hola(persona):  
    print("hola", persona)
```

Que quan es criden han de prendre un **valor**:

```
hola("jordi")
```

```
hola jordi
```

Cridem la funció
amb el
paràmetre amb
valor "jordi"

Un programa en Python

Els programes es solen posar en una funció anomenada main.

Línies de comentari

```
def main():  
    '''  
    Comportament caòtic  
    '''  
    print("Aquest programa implementa un comportament caòtic")  
    x = input("Entra un nombre entre 0 i 1")  
    x = float(x)  
    for i in range(10):  
        x = 3.9 * x * (1-x)  
        print(x)  
  
main()
```

x és una **variable** i serveix per donar nom a un valor i així poder cridar-lo quan ens interessi.

Això és una instrucció de tipus iteració.

Això és una instrucció input, i el que fa és escriure-ho i esperar una resposta acabada per <Enter>

Aquesta línia crida la funció.

Aquest és el cos de la instrucció iteració. La primera instrucció és una **assignació**.

Un altre programa en Python

```
def c2f(c):  
    '''  
    Conversió de temperatures Celsius a Fahrenheit  
    '''  
    f = 9.0 / 5.0 * c + 32  
    print("La temperature és:", f)
```

```
c2f(32)
```

La temperature és: 89.6

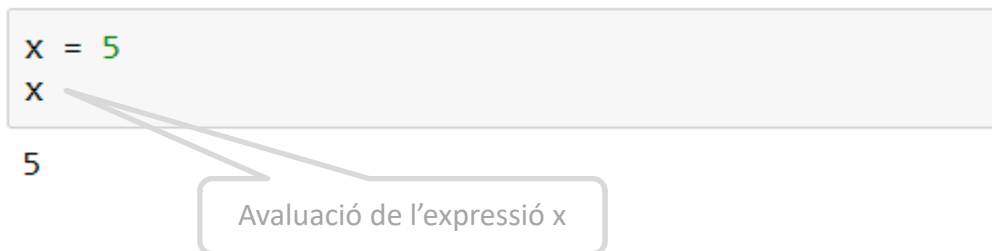
Un programa en Python

Els elements més importants que tenim per a construir un programa Python són:

•**Noms.** Els fem servir per anomenar les funcions i les variables. Tècnicament s'anomenen **identificadors**. Han de començar per lletra o `_` que pot ser seguit per qualsevol seqüència de lletres, dígitos o subratllats (no espais!). Són sensibles a majúscules i minúscules. Hi ha noms reservats.

and	del	for	is	raise
assert	elif	from	lambda	return
break	else	global	not	try
class	except	if	or	while
continue	exec	import	pass	yield
def	finally	in	print	

•**Expressions (i).** Són la part de codi que calcula o produeix nous valors de les dades. L'expressió més simple s'anomena **literal**, i s'usa per especificar un valor. Hem vist literals numèrics. Un identificador simple també pot ser una expressió (el nom d'una variable).



Un programa en Python

Python no pot avaluar
l'expressió `y` i genera un
error.

```
x = 5
```

```
print(y)
```

```
-----  
NameError                                Traceback (most recent call last)  
<ipython-input-50-36b2093251cd> in <module>()  
----> 1 print(y)  
  
NameError: name 'y' is not defined
```

Un programa en Python

- **Expressions (ii)**

Podem crear expressions combinant expressions més simples amb **operadors**. Els espais no compten.

```
3.9 * x * (1 - x)
9.0 / 5.0 * celsius + 32
```

Els operadors matemàtics segueixen les precedències estàndard.

```
((x1 - x2) / 2*n) + (spam / k**3)
```

Un programa en Python

- **Sortides.**

Hi ha la funció **print**, amb els següents arguments:

```
print(value1, value2,..., sep=' ', end='\n')
```

```
x = 5  
print("x = ", x)
```

```
x = 5
```

```
print("\n x = \n", x)
```

```
x =  
5
```

```
print("a","b",sep="")  
print(192,168,178,42,sep=".")
```

```
ab  
192.168.178.42
```

Un programa en Python

- **Assignacions (i).**

Assignacions simples: `<variable> = <expr>`

```
x = 3.9 * x * (1 - x)
fahrenheit = 9.0 / 5.0 * celsius + 32
x = 5
```

Assignacions d'entrada: `<variable> = input(<prompt>)`

```
x = input("Please enter a number between 0 and 1: ")
celsius = input("What is the Celsius temperature? ")
```

A més a més de nombres, podem entrar qualsevol expressió i avaluar-la:

```
a = eval(input("Entrada:"))
a
```

Entrada:3+4+5

12

Una variable es pot assignar
tants cops com faci falta.

```
a = 0
a = 1
a = a * 2
a = 0
a
```

0

Un programa en Python

- **Assignacions (ii)**

Assignacions simultànies:

```
<var>, <var>, ..., <var> = <expr>, <expr>, ..., <expr>
```

com per exemple

```
sum, diff = x+y, x-y
```

Aquest tipus d'assignació pot ser molt útil, com per exemple per intercanviar els valors de dues variables. Això no funciona!:

```
x = 3
y = 4
x = y
y = x
print(x,y)
```

4 4

Un programa en Python

- **Assignacions (iii)**

Assignacions simultànies:

Ho podríem fer amb una tercera variable:

```
x = 3
temp = x
y = 4
x = y
y = temp
print(x,y)
```

4 3

o amb la forma correcta de Python:

`x, y = y, x`

que ens permet escriure programes tant elegants com:

```
def main():
    """
    Càlcul de la mitja de dos examens
    """
    nota1, nota2 = eval(input("Entra les dues notes separades una coma: "))
    mitja = (nota1 + nota2) / 2.0
    print(mitja)
```

```
main()
```

```
Entra les dues notes separades una coma: 2,3
2.5
```

Un programa en Python

- **Iteracions (*loops*) definides.**

Es fa un nombre definit de vegades, i són el tipus més simple d'iteració.

```
for i in range(10):  
    print(i,end=" ")
```

0 1 2 3 4 5 6 7 8 9

En aquest cas tenim una iteració de tipus for

```
for <var> in <sequence>:  
    <body>
```

La <var> es diu **índex**.

La Identació és important!

Exemples:

```
for i in [0,1,2,3]:  
    print(i*i,end=" ")
```

0 1 4 9

Els nombres i Python

Les dades que un programa pot manipular i emmagatzemar són de diferents tipus. El **tipus** de la dada determina quins valors pot tenir i quines operacions es poden fer.

```
type(3), type(3.14)
```

```
(int, float)
```

```
x = -32  
type(x)
```

```
int
```

Els nombres i Python

```
print(3+4, 3+4.0)
```

7 7.0

```
print(10.0/3, 10/3)
```

3.3333333333333335 3.3333333333333335

```
print(10%3)
```

1

```
abs(-5)
```

5

operator	operation
+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation
%	remainder
abs ()	absolute value

Table 3.1: Python built-in numeric operations.

Els nombres i Python

Python també ens dona funcions matemàtiques dins d'una **biblioteca** (*library*) especial. Una biblioteca no és res més que un mòdul que conté definicions útils.

```
import math

def main():

    a,b,c = eval(input("Enter a, b and c separated by commas: "))
    d = b**2-4*a*c # discriminant

    if d < 0:
        print("This equation has no real solution")
    elif d == 0:
        x = (-b+math.sqrt(b**2-4*a*c))/2*a
        print("This equation has one solutions: ", x)
    else:
        x1 = (-b+math.sqrt(b**2-4*a*c))/2*a
        x2 = (-b-math.sqrt(b**2-4*a*c))/2*a
        print("This equation has two solutions: ", x1, " and", x2)
```

Els nombres i Python

Anem a escriure la funció **factorial** d'un nombre... (o el que és el mateix, el nombre de maneres diferents d'ordenar n coses)

$$n! = n(n-1)(n-2)\dots(1)$$

Per fer-ho només ens hem d'adonar que necessitem un **acumulador**, una eina molt útil per a programar.

```
fact = 1
for factor in [6,5,4,3,2,1]:
    fact = fact * factor
print(fact)
```

720

Els nombres i Python

Però això ens obliga a escriure una llista que pot ser molt llarga...
Aprofitem el tipus range:

```
range(stop)  
range(start, stop[, step])
```

```
>>> list(range(10))  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
>>> list(range(1, 11))  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
>>> list(range(0, 30, 5))  
[0, 5, 10, 15, 20, 25]  
>>> list(range(0, 10, 3))  
[0, 3, 6, 9]  
>>> list(range(0, -10, -1))  
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]  
>>> list(range(0))  
[]  
>>> list(range(1, 0))  
[]
```

```
>>> r = range(0, 20, 2)  
>>> r  
range(0, 20, 2)  
>>> 11 in r  
False  
>>> 10 in r  
True  
>>> r.index(10)  
5  
>>> r[5]  
10  
>>> r[:5]  
range(0, 10, 2)  
>>> r[-1]  
18
```

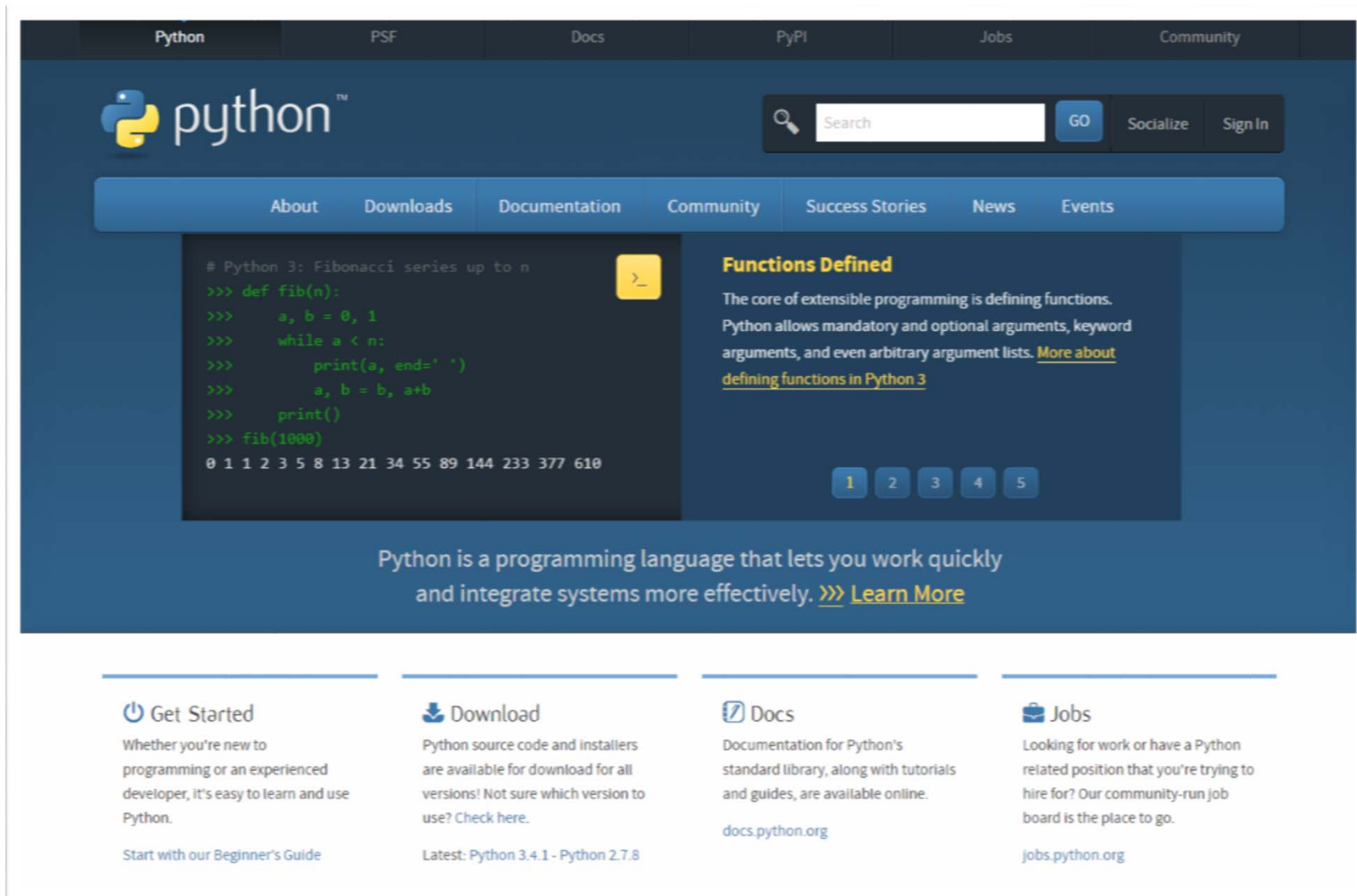

Els nombres i Python

```
def factorial(num):  
    factorial = 1  
    if num < 0:  
        print("Please enter a positive integer")  
    elif num == 0:  
        print("The factorial of 0 is 1")  
    else:  
        for i in range(1, num + 1):  
            factorial *= i  
    print("The factorial of", num, "is", factorial)
```

```
factorial(10)
```

The factorial of 10 is 3628800

<https://www.python.org/>



The image is a screenshot of the Python.org homepage. At the top, there is a dark blue navigation bar with links for Python, PSF, Docs, PyPI, Jobs, and Community. Below this is a large blue banner featuring the Python logo on the left. To the right of the logo is a search bar with a magnifying glass icon and a 'GO' button, followed by 'Socialize' and 'Sign In' links. Below the search bar is a horizontal menu with links: About, Downloads, Documentation, Community, Success Stories, News, and Events. The main content area is divided into two columns. The left column contains a code snippet for a Fibonacci function in Python 3, with a yellow terminal icon to its right. The right column is titled 'Functions Defined' and contains text about defining functions, with a link to 'More about defining functions in Python 3'. Below this text are five numbered buttons (1-5). At the bottom of the banner, a text line reads 'Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)'. Below the banner, there are four columns of links: 'Get Started' (with a power icon), 'Download' (with a download icon), 'Docs' (with a document icon), and 'Jobs' (with a briefcase icon). Each column contains a brief description and a link to a specific page.

Python PSF Docs PyPI Jobs Community

python™

Search GO Socialize Sign In

About Downloads Documentation Community Success Stories News Events

```
# Python 3: Fibonacci series up to n
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610
```

Functions Defined

The core of extensible programming is defining functions. Python allows mandatory and optional arguments, keyword arguments, and even arbitrary argument lists. [More about defining functions in Python 3](#)

1 2 3 4 5

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)


Get Started
Whether you're new to programming or an experienced developer, it's easy to learn and use Python.
[Start with our Beginner's Guide](#)

Download
Python source code and installers are available for download for all versions! Not sure which version to use? [Check here.](#)
Latest: Python 3.4.1 - Python 2.7.8

Docs
Documentation for Python's standard library, along with tutorials and guides, are available online.
docs.python.org

Jobs
Looking for work or have a Python related position that you're trying to hire for? Our community-run job board is the place to go.
jobs.python.org

<https://docs.python.org/3/>

 Python » English ▼ 3.6.2 ▼ Documentation »

| [modules](#) | [index](#)

Download

Download these documents

Docs for other versions

- [Python 2.7 \(stable\)](#)
- [Python 3.5 \(stable\)](#)
- [Python 3.7 \(in development\)](#)
- [Old versions](#)

Other resources

- [PEP Index](#)
- [Beginner's Guide](#)
- [Book List](#)
- [Audio/Visual Talks](#)

Python 3.6.2 documentation

Welcome! This is the documentation for Python 3.6.2.

Parts of the documentation:

What's new in Python 3.6?

or all "What's new" documents since 2.0

Tutorial

start here

Library Reference

keep this under your pillow

Language Reference

describes syntax and language elements

Python Setup and Usage

how to use Python on different platforms

Python HOWTOs

in-depth documents on specific topics

Installing Python Modules

installing from the Python Package Index & other sources

Distributing Python Modules

publishing modules for installation by others

Extending and Embedding

tutorial for C/C++ programmers

Python/C API

reference for C/C++ programmers

FAQs

frequently asked questions (with answers!)