

Turing Machine by Tom Dunne
American Scientist, March-April 2002

Algorísmica

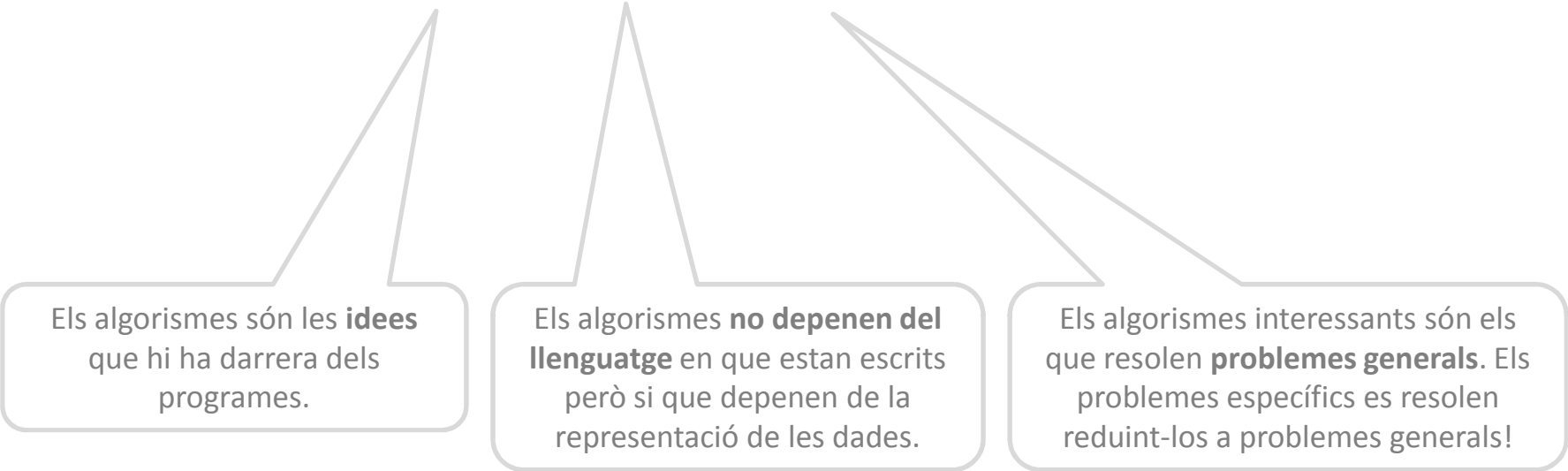
Què és un algorisme?

Jordi Vitrià

Què és algorisme?

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

Definició II: 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 dels programes.

Els algorismes **no depenen del llenguatge** en que estan escrits però si 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.

Attack Angle

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

Support Structure

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

Practice Schedule

4. 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. Swivel plastic bumpers sideways so top of bucket rim will fit against toilet seat.
5. adjusted toilet seat
- 6.

Exemple computacional (el problema de l'ordenació)

Input:

Una seqüència de nombres $\langle a_1..a_N \rangle$

Output:

La permutació (reordenació) $\langle a'_1..a'_N \rangle$ de la seqüència d'entrada de manera que $a'_1 < a'_2 < \dots < a'_N$

Observació:

Volem una solució correcta i eficient!

3, 6, 3, 6, 8, 5, 7, 9 \longrightarrow 3, 3, 5, 6, 6, 7, 8, 9

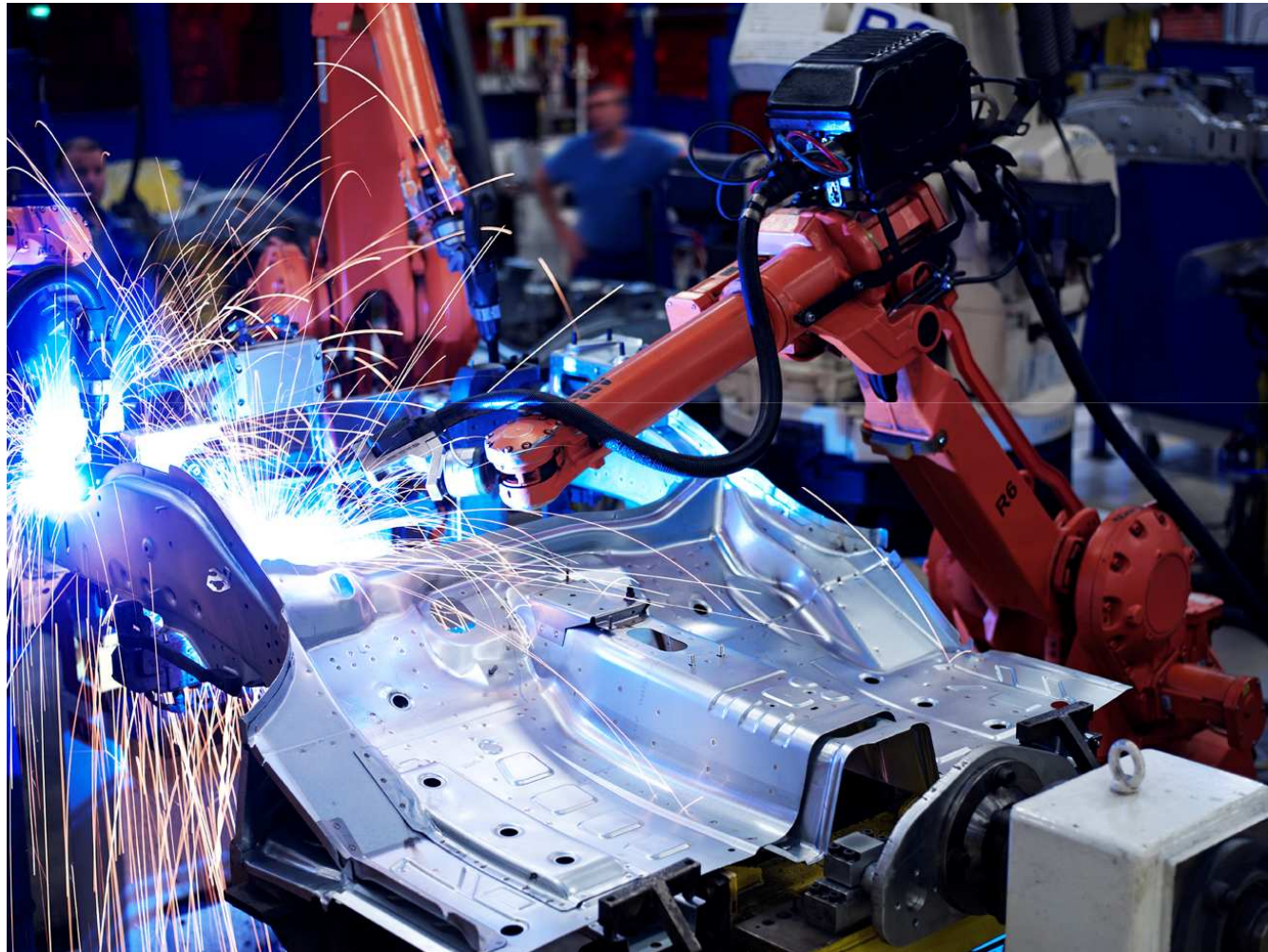
Correcció i Eficiència Algorísmica

Un algorisme es **correcte** si podem **demostrar** que retorna la sortida desitjada per a qualsevol entrada legal (per el problema de l'ordenació, això ha d'incloure entrades ja ordenades o entrades amb elements repetits!).

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

Això és fàcil per alguns algorismes, difícil per la majoria i fins i tot impossible per alguns.

Exemple



Exemple

TSP: Traveling
Salesman Problem.

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

HELP! WE'RE LOST!

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

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.

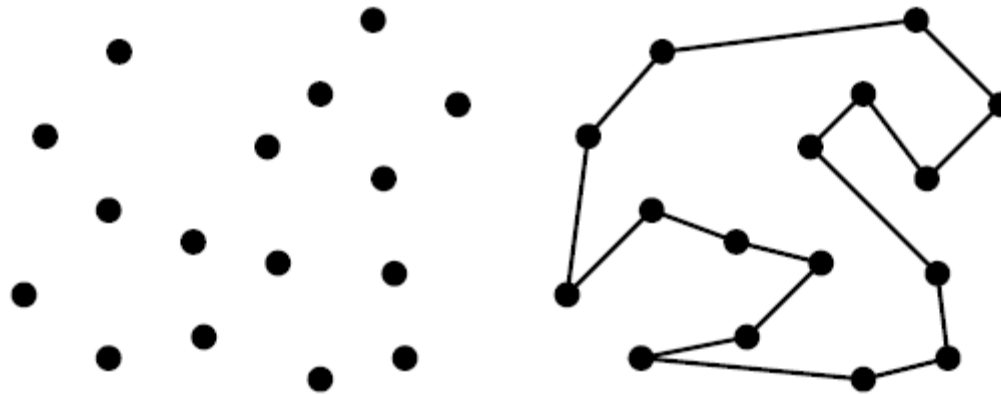
HERE'S THE CORRECT START...
Begin at Chicago, Illinois. From there, lines show correct route as far as Erie, Pennsylvania. Next, do you go to Carlisle, Pennsylvania or Wana, West Virginia? Check the easy instructions on back of this entry blank for details.

© PROCTER & GAMBLE 1962

OFFICIAL RULES ON REVERSE SIDE

Exemple (i)

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



Exemple (ii)

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

$i = i + 1$

Identació

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

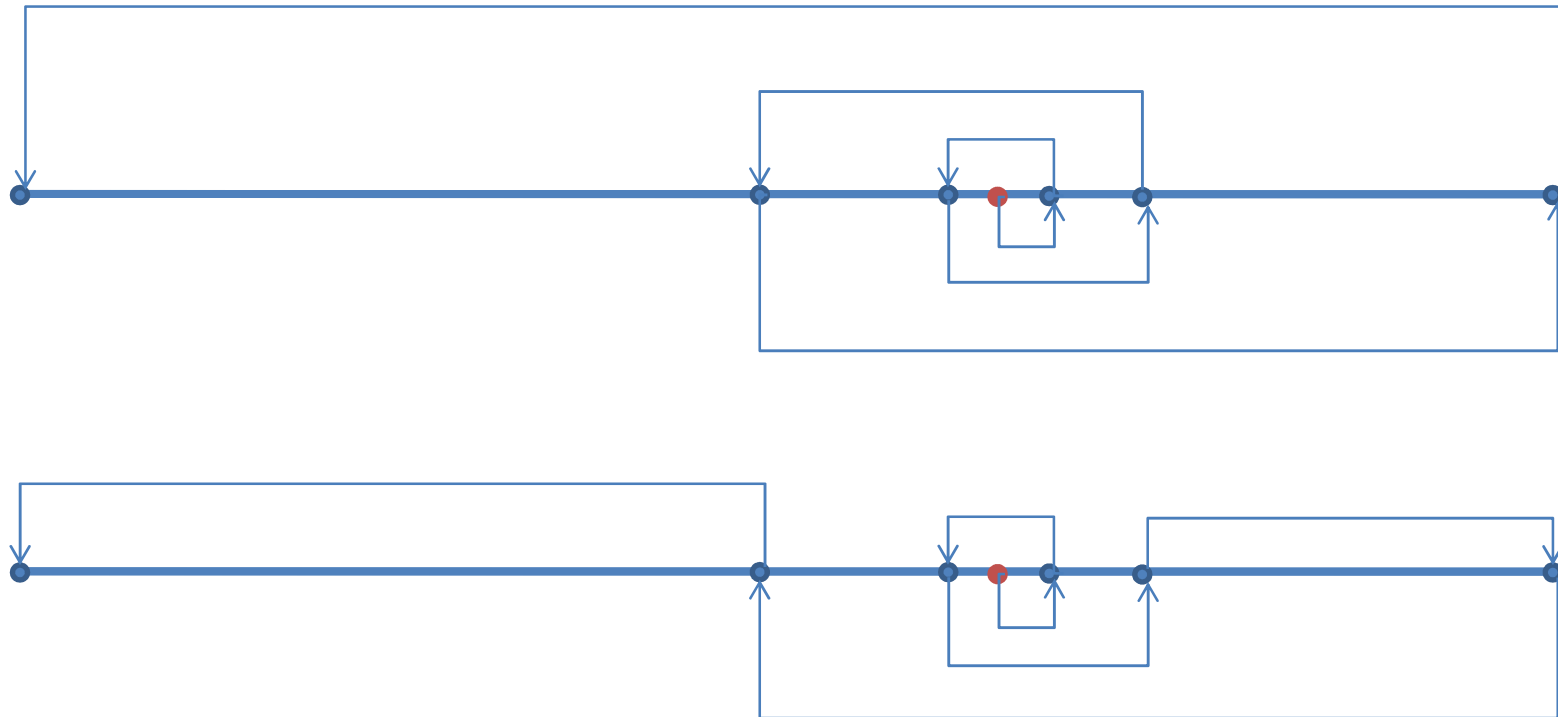
Visitem p_i

Retorna la seqüència

Algorisme especificat en
pseudocodi

Exemple (iii)

És evident que no és correcte!



Exemple (iv)

Solució II: Considerem tots el possibles passos parcials 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!

Example (v)

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
23758251185210916864000000000000000000000000000
```

Example

Applegate, Bixby, Chvátal, Cook & Helsingaun van trobar al 2004 la solució òptima per recórrer 24,978 ciutats de Suècia.



Exemple

Solució aproximada per
100.000 punts que
representen la Mona Lisa.



Com expressem els algorismes?

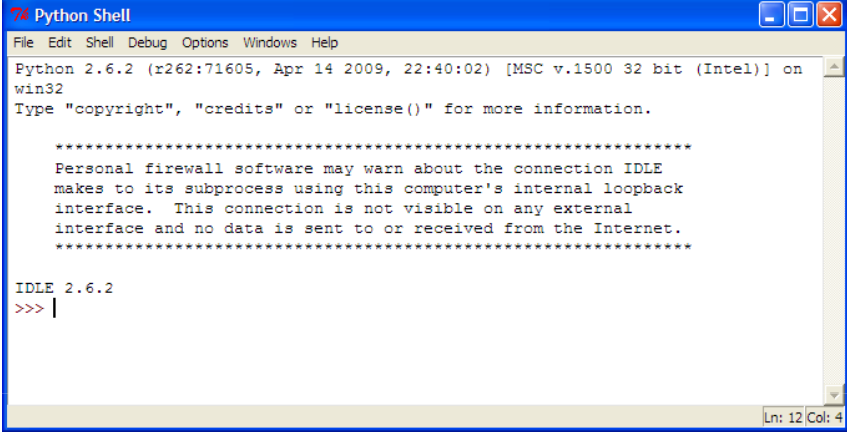
Fins ara hem usat el pseudocodi, però també podem usar un llenguatge d'alt nivell, **Python**, molt proper al pseudocodi, que ens permetrà executar els algorismes!

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

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

Un programa en Python

Si cridem a l'interpret de Python apareix:



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.6.2 (r262:71605, Apr 14 2009, 22:40:02) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

IDLE 2.6.2
>>> |
```

i ja podem començar a donar ordres...

```
>>> print "Hello, World"
Hello, World
>>> print 2 + 3
5
>>> print "2 + 3 =", 2 + 3
2 + 3 = 5
```


→ *instrucció/statement*

Un programa en Python

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

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

```
>>> def hello():  
    print "Hello"  
    print "Computers are Fun"
```



>>>

Línia en blanc per
indicar que és el final
de la definició

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

```
>>> hello()  
Hello  
Computers are Fun  
>>>
```

Un programa en Python

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

```
>>> def greet(person):  
    print "Hello", person  
    print "How are you?"
```

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

```
>>> greet("John")  
Hello John  
How are you?  
>>> greet("Emily")  
Hello Emily  
How are you?  
>>>
```



Cridem la funció amb
el paràmetre amb valor
"John"

Un programa en Python

Podem crear-los amb un editor de text o amb un entorn de programació.

Si volem cridar la funció en el futur, l'hem de guardar. Podem crear un programa escrivint les funcions en un fitxer apart, que anomenarem **mòdul** o **script**, que guardarem al disc.

Escrivim i guardem al disc amb el nom `chaos.py`:

L'extensió `.py` indica que és un mòdul Python.

Comentaris que no formen part del programa

```
# File: chaos.py
# A simple program illustrating chaotic behavior.

def main():
    print "This program illustrates a chaotic function"
    x = input("Enter a number between 0 and 1: ")
    for i in range(10):
        x = 3.9 * x * (1 - x)
        print x

main()
```

Els programes es solen posar en una funció anomenada `main`.

Un programa en Python

Hi ha diverses maneres de cridar el programa, però la més bàsica és aquesta:

```
>>> import chaos
This program illustrates a chaotic function
Enter a number between 0 and 1: .25
0.73125
0.76644140625
0.698135010439
0.82189581879
0.570894019197
0.955398748364
0.166186721954
0.540417912062
0.9686289303
0.118509010176
>>>
```

Aquesta línia indica a l'interpret que volem carregar el mòdul chaos del fitxer chaos.py i a continuació l'executa.

Quan importem un mòdul d'aquesta manera, l'interpret crea un fitxer intermedi anomenat chaos.pyc (i que conté el *byte code*) que fa servir en el procés d'interpretació/compilació. Els fitxers .pyc poden servir per anar més ràpid en el futur...

El mòdul resta carregat durant tota la **sessió**, i podem cridar-lo amb la instrucció `chaos.main()`

Si no haguéssim creat la funció `main` no podríem!

Si la volem recarregar (per exemple, perquè l'hem editat en paral·lel): `reload(chaos)`

Python té una llista de llocs on espera trobar la funció (començant pel directori des d'on l'hem cridat: C:\Python26). Per fer-ho des d'un altre lloc se li ha de dir!

```
>>> import sys
>>> sys.path.append("C:/")
```


Un programa en Python

Línies de comentari

```
# File: chaos.py  
# A simple program illustrating chaotic behavior.
```

```
def main():
```

Els programes es solen posar en una funció anomenada main.

```
    print "This program illustrates a chaotic function"
```

```
    x = input("Enter a number between 0 and 1: ")
```

```
    for i in range(10):
```

```
        x = 3.9 * x * (1 - x)
```

```
    print x
```

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

main()

Aquesta línia crida la funció.

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

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>

Un programa en Python

```
# convert.py
#     A program to convert Celsius temps to Fahrenheit
# by: Suzie Programmer

def main():
    celsius = input("What is the Celsius temperature? ")
    fahrenheit = 9.0 / 5.0 * celsius + 32
    print "The temperature is", fahrenheit, "degrees Fahrenheit."

main()
```

Un programa en Python

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

- **Noms.** Els fem servir per anomenar els mòduls, 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 *underscores* (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).

```
>>> x = 5
>>> x
5
>>> print x
5
```

Avaluació de l'expressió x

```
>>> print spam
Traceback (innermost last):
  File "<pyshell#34>", line 1, in ?
    print spam
NameError: spam
>>>
```

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

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)
```

- **Sortides.**

Descriurem la sintaxi d'aquestes instruccions amb un **metallenguatge**:

```
print
print <expr>
print <expr>, <expr>, ..., <expr>
print <expr>, <expr>, ..., <expr>,
```

Afegeix un espai, però no salta línia.

```
print 3+4
print 3, 4, 3 + 4
print
print 3, 4,
print 3+4
print "The answer is", 3 + 4
```

Aquí usem un literal alfanumèric: **string**.

```
7
3 4 7

3 4 7
The answer is 7
```

Un programa en Python

Una variable es pot assignar
tants cops com faci falta.

- **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? ")
```

```
>>> myVar = 0
>>> myVar
0
>>> myVar = 7
>>> myVar
7
>>> myVar = myVar + 1
>>> myVar
8
```

A més a més de nombres, podem entrar qualsevol expressió!:

```
>>> ans = input("Enter an expression: ")
Enter an expression: 3 + 4 * 5
>>> print ans
23
>>>
```

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 = y
y = x
```

```
# variables      x  y
# initial values 2  4
x = y
# now            4  4
y = x
# final          4  4
```


Un programa en Python

- **Assignacions (iii)**

Assignacions simultànies:

Ho podríem fer amb una tercera variable:

```
temp = x
x = y
y = temp
```

# variables	x	y	temp
# initial values	2	4	no value yet
temp = x			
#	2	4	2
x = y			
#	4	4	2
y = temp			
#	4	2	2

o amb la forma correcta de Python:

```
x, y = y, x
```

que ens permet escriure programes tant elegants com:

```
def main():
    print "This program computes the average of two exam scores."

    score1, score2 = input("Enter two scores separated by a comma: ")
    average = (score1 + score2) / 2.0

    print "The average of the scores is:", average

main()
```

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):  
    x = 3.9 * x * (1 - x)  
    print x
```

```
>>> range(10)  
[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
```

0
1
2
3

```
>>> for odd in [1, 3, 5, 7, 9]:  
    print odd * odd
```

1
9
25
49
81

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 'int'>
>>> type(3.14)
<type 'float'>
>>> type(3.0)
<type 'float'>
>>> myInt = -32
>>> type(myInt)
<type 'int'>
>>> myFloat = 32.0
>>> type(myFloat)
<type 'float'>
```

Els nombres i Python

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

Table 3.1: Python built-in numeric operations.

```
>>> 3.0 + 4.0
7.0
>>> 3 + 4
7
>>> 3.0 * 4.0
12.0
>>> 3 * 4
12
>>> 10.0 / 3.0
3.333333333333333
>>> 10 / 3
3
>>> 10 % 3
1
>>> abs(5)
5
>>> abs(-3.5)
3.5
```

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.

$$ax^2 + bx + c = 0$$
$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
# quadratic.py
#   A program that computes the real roots of a quadratic equation.
#   Illustrates use of the math library.
#   Note: this program crashes if the equation has no real roots.

import math # Makes the math library available.

def main():
    print "This program finds the real solutions to a quadratic"
    print

    a, b, c = input("Please enter the coefficients (a, b, c): ")

    discRoot = math.sqrt(b * b - 4 * a * c)
    root1 = (-b + discRoot) / (2 * a)
    root2 = (-b - discRoot) / (2 * a)

    print
    print "The solutions are:", root1, root2

main()
```

Els nombres i Python

This program finds the real solutions to a quadratic

Please enter the coefficients (a, b, c): 3, 4, -2

The solutions are: 0.387425886723 -1.72075922006

This program finds the real solutions to a quadratic

Please enter the coefficients (a, b, c): 1, 2, 3

Traceback (innermost last):

File "<stdin>", line 1, in ?

File "quadratic.py", line 13, in ?

discRoot = math.sqrt(b * b - 4 * a * c)

OverflowError: math range error

Hi ha un error perquè

$$b^2 - 4 * a * c < 0$$

i la funció `sqrt` no pot
calcular l'arrel quadrada
d'un nombre negatiu.

Els nombres i Python

Python	Mathematics	English
<code>pi</code>	π	An approximation of pi.
<code>e</code>	e	An approximation of e .
<code>sin(x)</code>	$\sin x$	The sine of x .
<code>cos(x)</code>	$\cos x$	The cosine of x .
<code>tan(x)</code>	$\tan x$	The tangent of x .
<code>asin(x)</code>	$\arcsin x$	The inverse of sine x .
<code>acos(x)</code>	$\arccos x$	The inverse of cosine x .
<code>atan(x)</code>	$\arctan x$	The inverse of tangent x .
<code>log(x)</code>	$\ln x$	The natural (base e) logarithm of x
<code>log10(x)</code>	$\log_{10} x$	The common (base 10) logarithm of x .
<code>exp(x)</code>	e^x	The exponential of x .
<code>ceil(x)</code>	$\lceil x \rceil$	The smallest whole number $\geq x$
<code>floor(x)</code>	$\lfloor x \rfloor$	The largest whole number $\leq x$

Table 3.2: Some math library functions.

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.

```
Initialize the accumulator variable
Loop until final result is reached
    update the value of accumulator variable
```

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

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

Els nombres i Python

Però això ens obliga a escriure una llista molt llarga...

Aprofitem la funció `range`:

```
>>> range(10)                                range(n)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> range(5,10)                              range(start,n)
[5, 6, 7, 8, 9]

>>> range(5, 10, 3)                          range(start, n, step)
[5, 8]
```

```
Llavors: # factorial.py
#         Program to compute the factorial of a number
#         Illustrates for loop with an accumulator

def main():
    n = input("Please enter a whole number: ")
    fact = 1
    for factor in range(n,1,-1):
        fact = fact * factor
    print "The factorial of", n, "is", fact

main()
```

Els nombres i Python

```
>>> factorial.main()
Please enter a whole number: 10
The factorial of 10 is 3628800

>>> factorial.main()
Please enter a whole number: 13
Traceback (innermost last):
  File "<stdin>", line 1, in ?
  File "factorial.py", line 9, in main
    fact = fact * factor
OverflowError: integer multiplication
```

Els diferents tipus tenen diferents representacions, que depenen del nombre de bits que s'usen per emmagatzemar-los a l'ordinador. El sencer es representen amb 32 bits, i per tant, el sencer més gran que es pot representar és el 2.147.483.647.

Els nombres i Python

Els flotants ens permeten representar nombres més grans, però amb menys precisió:

```
Please enter a whole number. 15
The factorial of 15 is 1.307674368e+12
```

Python ens dona una segona opció: el long int, que només tenen com a límit la memòria de l'ordinador:

[illegible]

Els nombres i Python

```
# factorial2.py

def main():
    n = input("Please enter a whole number: ")
    fact = 1L      # Use a long int here
    for factor in range(n,0,-1):
        fact = fact * factor
    print "The factorial of", n, "is", fact

>>> import factorial2
Please enter a whole number: 13
The factorial of 13 is 6227020800

>>> factorial2.main()
Please enter a whole number: 100
The factorial of 100 is 933262154439441526816992388562667004907159682
643816214685929638952175999932299156089414639761565182862536979208272
237582511852109168640000000000000000000000000000
```

Els nombres i Python

Què passa quan barregem dos tipus diferents en una mateixa expressió?
Podem deixar que Python ho converteixi automàticament (a vegades és perillós):

```
average = sum / n
```

O aplicar la conversió explícitament:

```
average = float(sum) / n
```

```
>>> int(4.5)
4
>>> int(3.9)
3
>>> long(3.9)
3L
>>> float(int(3.3))
3.0
>>> int(float(3.3))
3
>>> int(float(3))
3
```

```
>>> round(3.14)
3.0
>>> round(-3.14)
-3.0
>>> round(3.5)
4.0
>>> round(-3.5)
-4.0
>>> int(round(-3.14))
-3
```

