



UNIVERSITAT DE
BARCELONA

Grau d'Enginyeria Informàtica
Facultat de Matemàtiques i Informàtica



Algorísmica

Introducció als algorismes II

Mireia Ribera & Jordi Vitrià

Strings i Python

Un *string* és una seqüència de caràcters, que es pot emmagatzemar en variables:

```
str = "Hola"  
str2 = 'spam'  
print(str, str2)
```

Hola spam

```
type(str2)
```

str

Podem entrar *strings* des del teclat:

```
nom = input("Quin és el teu nom?")
```

Quin és el teu nom?

De fet **tot el que entra pel teclat és un *string*.**

Strings i Python

Si volem entrar un altre tipus ho hem de fer així:

```
edat = eval(input("Quina és la teva edat? "))
```

```
Quina és la teva edat? 54
```

```
type(edat)
```

```
int
```

El que hem fet és dir-li que volem que ho interpreti com una expressió de Python!

```
x = 45
```

```
res = eval(input("Quina expressió vols calcular? "))
```

```
Quina expressió vols calcular? x + 67
```

```
res
```

```
112
```

Strings i Python

Per accedir als elements d'un *string* hem de veure com Python els indexa:

H	e	l	l	o		B	o	b
0	1	2	3	4	5	6	7	8

Llavors podem accedir als valors de cada element de la seqüència o fins i tot a subseqüències:

```
s = "Hello Bob"
```

```
x = 8  
print(s[0], s[2], s[x-2])
```

```
H l B
```

Strings i Python

Slicing

H	e	l	l	o		B	o	b
0	1	2	3	4	5	6	7	8

```
s[0:3]
```

```
'Hel'
```

```
s[5:9]
```

```
' Bob'
```

```
s[:3]
```

```
'Hel'
```

```
s[3:]
```

```
'lo Bob'
```

```
s[:]
```

```
'Hello Bob'
```

Strings i Python

També podem **concatenar** (+) i **repetir** (*), o demanar la seva llargada:

```
"Bread" + " and " + "Breakfast"
```

```
'Bread and Breakfast'
```

```
"Bread" + " & " * 3 + "Breakfast"
```

```
'Bread & & & Breakfast'
```

```
len("Bread" + " & " * 3 + "Breakfast")
```

23

Operator	Meaning
+	Concatenation
*	Repetition
<string>[]	Indexing
len(<string>)	length
<string>[:]	slicing

Table 4.1: Python string operations

Strings i Python

```
def mes():  
    mesos = "GenFebMarAbrMaiJunJulAgoSetOctNovDes"  
    n = eval(input("Quin mes vols? "))  
    pos = (n-1) * 3  
    m = mesos[pos:pos+3]  
    print("L'abreviació és: ", m)
```

```
mes()
```

```
Quin mes vols? 4  
L'abreviació és: Abr
```

Strings i Python

L'ordinador emmagatzema els caràcters de forma numèrica.

Una forma estàndard s'anomena codificació ASCII (*American Standard Code for Information Interchange*), però tal i com el nom indica, no considera els caràcters que no s'usen en l'anglès. Usa 7 bits per caràcter.

Per això hi ha el sistema *Unicode*, que considera els caràcters de totes les llengües. Usa 16 bits per caràcter. Per compatibilitat, és un superconjunt de l'ASCII.

Python ens dona funcions per accedir a aquests codis:

```
ord("a")
```

97

```
ord("A")
```

65

```
ord("à")
```

224

```
chr(97)
```

'a'

```
chr(90)
```

'Z'

ASCII: American Standard Code for Information Interchange

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Source: www.LookupTables.com

Strings i Python

```
def main():
    """
    text2numbers.py
    A program to convert a textual message into a sequence of
    numbers, using the underlying ASCII encoding
    """

    # Get the message to encode
    message = input("Please enter the message to encode: ")
    print("\nHere are the ASCII codes:")

    # Loop through the message and print out the ASCII values
    for ch in message:
        print(ord(ch), " ", end="") # use end="" to print all on one line
    print

main()
```

Please enter the message to encode: Hola

Here are the ASCII codes:

72 111 108 97

El `for` itera
sobre strings,
tuples o llistes

Strings i Python

```
def main():
    """
    numbers2text.py
    A program to convert a sequence of ASCII numbers into
    a string of text
    """
    # Get the message to encode
    instring = input("Please enter the ASCII encoded message: ")
    print
    # Loop through each substring and build the ASCII message
    message = ""
    for numStr in instring.split():      Mètode sobre un string
        asciiNum = eval(numStr)
        message += chr(asciiNum)
    print
    print("The decoded message is:", message)

main()
```

Please enter the ASCII encoded message: 72 111 108 97
The decoded message is: Hola

Strings i Python

split

Amb `split` puc separar una cadena en una llista amb els diferents trossets, indicant el separador.

```
# Exemple
cadena="456.342.120€"
llista=cadena.split('.')
print(llista)
```

```
['456', '342', '120€']
```

Si no indico un separador, per defecte és l'espai.

```
# Exemple
cadena="El gos, i el gat, menjàven plegats."
llista = cadena.split()
print(llista)
```

```
['El', 'gos,', 'i', 'el', 'gat,', 'menjàven', 'plegats.']
```

Strings i Python: Altres funcions interessants

- **Join**

Un dels usos més estesos de join consisteix en convertir llistes a cadenes de text.

Exemple

```
llista = ['El', 'gos,', 'i', 'el', 'gat,', 'menjàven', 'plegats.']  
cadena = " ".join(llista) # l'espai farà de separador  
print(cadena)
```

El gos, i el gat, menjàven plegats.

- **Strip**

Elimina els caràcters indicats de l'inici i del final de la cadena, o els espais en blanc si no s'indiquen caràcters específics.

Exemple

```
"introducció a Python".strip('nio')
```

'troducció a Pyth'

" introducció ".strip()

'introducció'

- **islower, isupper i isalpha**

Verifiquen si la cadena és majúscules, minúscules o tota de lletres respectivament.

- **lower i upper**

Converteixen la cadena de majúscules a minúscules i de minúscules a majúscules respectivament.

Exemple

```
cadena1 = "ENFADAT!"  
print(cadena1, " is upper?", cadena1.isupper())  
print(cadena1, " is lower?", cadena1.islower())  
cadena2 = "123"  
cadena3 = "a10b"  
cadena4 = "abc"  
print(cadena2, " is alpha?", cadena2.isalpha())  
print(cadena3, " is alpha?", cadena3.isalpha())  
print(cadena4, " is alpha?", cadena4.isalpha())  
print(cadena1, " a lower:", cadena1.lower())  
cadena5 = "adormit"  
print(cadena5, " a uppper:", cadena5.upper())
```

ENFADAT! is upper? True

ENFADAT! is lower? False

123 is alpha? False

a10b is alpha? False

abc is alpha? True

ENFADAT! a lower: enfadat!

adormit a uppper: ADORMIT

Strings i Python (formatat)

Python a més permet formatar strings. A dins del text podem un "template" (patró) i al final del text posem els valors, ambdós precedits per %. En el cas dels float en el template podem indicar l'amplada que ocuparà el nombre (per fer que tots s'alineïn a la dreta, per exemple), i la precisió.

```
"Hola %s %s, sembla que ha guanyat $%d!" %("Sr.", "Fuster", 10000)
```

```
'Hola Sr. Fuster, sembla que ha guanyat $10000!'
```

```
"Aquest enter, %5d, s'ha col.locat en un camp d'amplada 5" %(7)
```

```
"Aquest enter,      7, s'ha col.locat en un camp d'amplada 5"
```

```
"Aquest enter, %10d, s'ha col.locat en un camp d'amplada 10" %(7)
```

```
"Aquest enter,           7, s'ha col.locat en un camp d'amplada 10"
```

```
"Aquest decimal, %10.5f, té una amplada de 10 i una precisió de 5" %(3.1415926)
```

```
'Aquest decimal,      3.14159, té una amplada de 10 i una precisió de 5'
```

```
"Aquest decimal, %0.5f, té una amplada de 0 i una precisió de 5" %(3.1415926)
```

```
'Aquest decimal, 3.14159, té una amplada de 0 i una precisió de 5'
```

```
"Compara %f i %0.20f" %(3.14, 3.14)
```

```
'Compara 3.140000 i 3.140000000000000012434'
```

Strings i Python

Suppose you are writing a computer system for a bank. Your customers would not be happy to learn that a check went through an amount “very close to \$107.56”. They want to know that the bank is keeping precise track for their money. Even though the amount of error in a given value is very small, the small errors can be compounded when doing lots of calculations, and the resulting error could add up to some real cash. That’s not a satisfactory way of doing business.

A better approach would be to make sure that our program used exact values to represent money. We can do that by keeping track of the money in cents and using an int to store it. We can then convert this into dollars and cents in the output step. If `total` represents the value in cents, then we can get the number of dollars by `total / 100` and the cents from `total % 100`. Both of these are integer calculations and, hence, will give exact results. Here is the program:

```
def main():
    print("Change counter")
    print
    print("Please enter the count of each coin type")
    quarters = eval(input("Quarters: "))
    dimes = eval(input("Dimes: "))
    nickels = eval(input("Nickels: "))
    pennies = eval(input("Pennies: "))
    total = quarters * 25 + dimes * 10 + nickels * 5 + pennies
    print
    print("The total value of your change is ${d:%02d}
          % (total/100, total % 100))
```

Funcions i Python

Fins ara hem escrit tots els programes en una única funció (`main()`).

Per diverses raons (economia a l'escriure, manteniment del software, disseny) val la pena fer servir diferents **funcions**. Una funció és un **subprograma**, o un programa dins del programa. Per tant no són res més que una seqüència d'instruccions amb un nom. Una funció es pot **cridar** des de qualsevol lloc del programa pel seu nom.

```
def main():
    print("Happy birthday to you!")
    print("Happy birthday to you!")
    print("Happy birthday, dear Fred.")
    print("Happy birthday to you!")

main()

def happy():
    print("Happy birthday to you!")

def singFred():
    happy()
    happy()
    print("Happy birthday, dear Fred.")
    happy()

singFred()

Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Fred.
Happy birthday to you!
```


Funcions i Python

O encara millor

```
def happy():  
    print("Happy birthday to you!")  
  
def sing(person):  
    happy()  
    happy()  
    print("Happy birthday, dear", person+".")  
    happy()  
  
def main():  
    sing("Fred")  
    print  
    sing("Lucy")  
    print  
    sing("Elmer")
```

Funcions i Python

Scope és el nom que donem als llocs d'un programa en els que una variable pot ser referida. Les variables dins d'una funció només es poden referir dins de la funció, són **locals**, i per això poden tenir el mateix nom que variables externes.

L'única manera que té una funció per veure les variables d'una altra funció és passar-li com a paràmetre.

La definició d'una funció és:

```
def <name>(<formal-parameters>):  
    <body>
```

El `<name>` és un identificador, i `<formal-paramters>` és una llista (buida) de variables (identificadors).

La funció es crida així: `<name>(<actual-parameters>)`

Funcions i Python

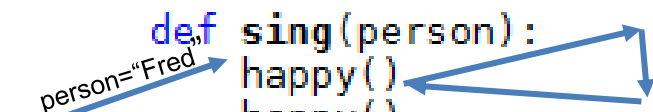
Quan Python rep la crida d'una funció, fa quatre coses:

1. El programa que fa la crida es suspèn/congela en el punt de la crida.
2. Els paràmetres de la funció s'assignen als valors de la crida.
3. S'executa el cos de la funció.
4. Retorna el control al punt de programa posterior a la crida.

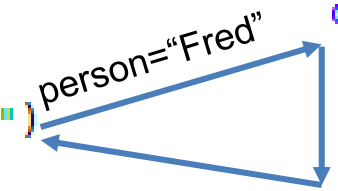
```
def main():  
    sing("Fred")  
    print  
    sing("Lucy")  
  
def sing(person):  
    happy()  
    happy()  
    print("Happy birthday, dear", person+".")  
    happy()  
  
    person="Fred"
```

Funcions i Python

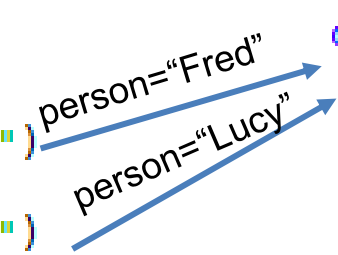
```
def main():  
    sing("Fred")  
    print  
    sing("Lucy")  
  
def sing(person):  
    happy()  
    happy()  
    print("Happy birthday, dear", person+".")  
    happy()  
  
def happy():  
    print("Happy birthday to you!")
```



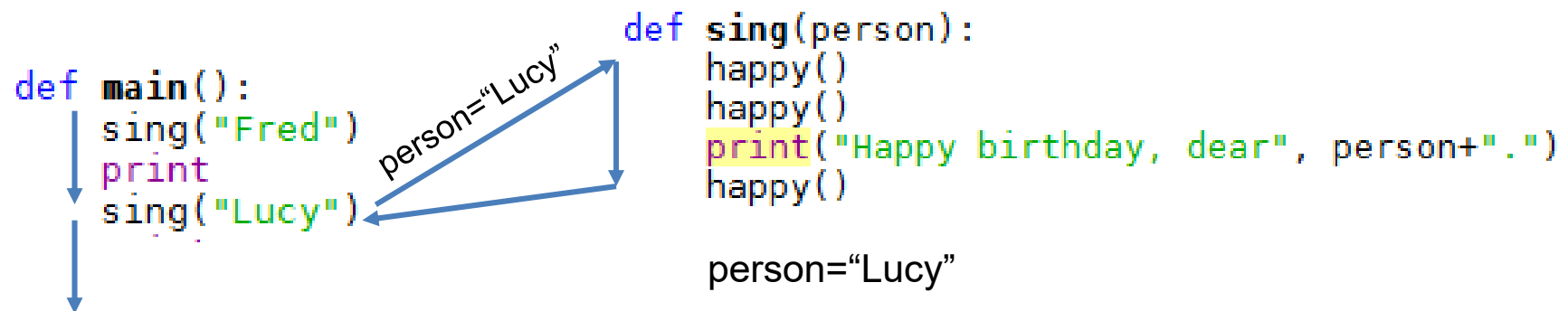
```
def main():  
    sing("Fred")  
    print  
    sing("Lucy")  
  
def sing(person):  
    happy()  
    happy()  
    print("Happy birthday, dear", person+".")  
    happy()
```



```
def main():  
    sing("Fred")  
    print  
    sing("Lucy")  
  
def sing(person):  
    happy()  
    happy()  
    print("Happy birthday, dear", person+".")  
    happy()  
  
    person="Fred"
```



Funcions i Python



Funcions i Python

A vegades també volem que les funcions ens retornin valors:

```
import math
discRt = math.sqrt(b*b - 4*a*c)
```

Que s'escriuen així:

```
def square(x):
    return x*x

def sumDiff(x,y):
    sum = x + y
    diff = x - y
    return sum, diff

num1, num2 = eval(input("Entra dos nombres (separats per ,) "))
s,d = sumDiff(num1,num2)
print("La suma és: ", s, "i la diferència és: ", d)
```

```
Entra dos nombres (separats per ,) 10, 5
La suma és:  15 i la diferència és:  5
```

Tècnicament, totes les funcions retornen alguna cosa al programa que les ha cridat, fins i tot les que no fan `return`! Aquestes retornen un objecte especial que s'anomena `none`.

Estructures de Control i Python

Fins ara, un programa és una seqüència pura d'instruccions, però a vegades necessitem trencar o alterar aquesta seqüència!

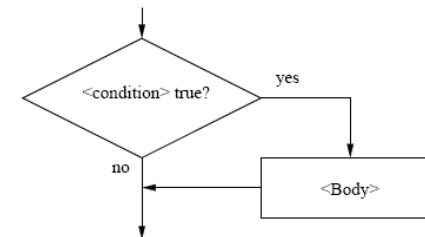
Això ho farem amb unes instruccions especials anomenades **estructures de control**.

```
"""
input the temperature in degrees Celsius (call it celsius)
Calculate fahrenheit as 9/5 celsius + 32
Output fahrenheit
"""

if fahrenheit > 90:
    print("a heat warning")
if fahrenheit < 30:
    print("a cold warning")
```

Estructures de Control i Python

```
if <condition>:  
    <body>
```



```
"""  
convert2.py  
    A program to convert Celsius temps to Fahrenheit.  
    This version issues heat and cold warnings.  
"""  
  
def main():  
    celsius = eval(input("What is the Celsius temperature?"))  
    fahrenheit = 9.0 / 5.0 * celsius + 32  
    print("The temperature is", fahrenheit, "degrees fahrenheit.")  
    # print warnings for extreme temps  
    if fahrenheit > 90:  
        print("It's really hot out there, be careful!")  
    if fahrenheit < 30:  
        print("Brrrrr. Be sure to dress warmly!")  
  
main()
```


Estructures de Control i Python

Les **condicions** tindran la forma

`<expr> <relop> <expr>`

On `<relop>` és un **operador relacional**.

Les condicions retornen un booleà (*True* o *False*).

Podem comparar nombres o *strings* (per ordre **lexicogràfic**).

Python	Mathematics	Meaning
<code><</code>	<code><</code>	Less than
<code><=</code>	<code>≤</code>	Less than or equal to
<code>==</code>	<code>=</code>	Equal to
<code>>=</code>	<code>≥</code>	Greater than or equal to
<code>></code>	<code>></code>	Greater than
<code>!=</code>	<code>≠</code>	Not equal to

```
>>> 3 < 4
1
>>> 3 * 4 < 3 + 4
0
>>> "hello" == "hello"
1
>>> "hello" < "hello"
0
>>> "Hello" < "hello"
1
```

Estructures de Control i Python

```
"""
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 = eval(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()
```

Aquest programa no funciona en algunes ocasions...

Estructures de Control i Python

Podríem posar-hi:

```
a, b, c = eval(input("Please enter the coefficients(a,b,c):"))
discrim = b * b - 4 * a * c
if discrim > 0:
    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)
```

EL seu funcionament seria:

```
>>> quadratic2.main()
This program finds the real solutions to a quadratic

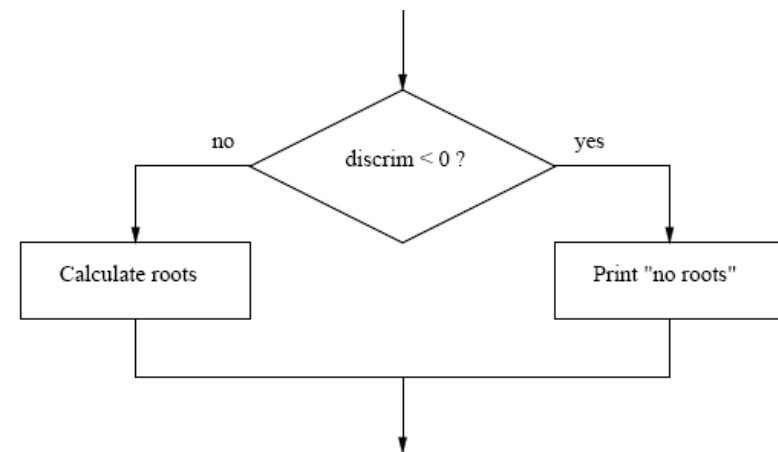
Please enter the coefficients (a, b, c): 1,2,3
>>>
```

Però no donem informació a l'usuari de què passa! Hem de fer alguna cosa quan no es pot aplicar.

Estructures de Control i Python

Això ho podem solucionar amb un nou tipus de decisió:

```
if <condition>:  
    <statements>  
else:  
    <statements>
```



Però de fet, el programa necessitaria encara més opcions!

```
when < 0: handle the case of no roots  
when = 0: handle the case of a double root  
when > 0: handle the case of two distinct roots.
```

Estructures de Control i Python

Ho podem fer així:

```
if discrim < 0:
    print("Equation has no real roots")
else:
    if discrim == 0:
        root = -b / (2 * a)
        print("There is a double root at", root)
    else:
        discRoot = math.sqrt(b * b - 4 * a * c)
```

O millor encara, així:

```
if <condition1>:
    <case1 statements>
elif <condition2>:
    <case2 statements>
elif <condition3>:
    <case3 statements>
...
else:
    <default statements>
```

```
if discrim < 0:
    print("Equation has no real roots")
elif discrim == 0:
    root = -b / (2 * a)
    print("There is a double root at", root)
else:
    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)
```

Estructures de Control i Python

Suposem que volem fer un programa que calculi el promig d'una seqüència arbitrària de nombres.

```
# average.py

def main():
    n = eval(input("How many numbers do you have? "))
    sum = 0.0
    for i in range(n):
        x = eval(input("Enter a number >> "))
        sum = sum + x
    print("The average of the numbers is", sum / n)
```

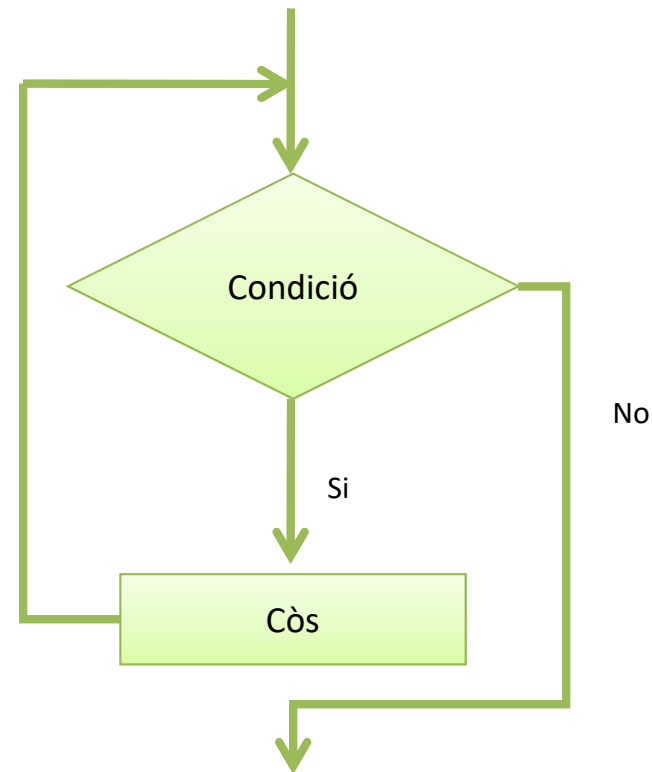
És correcte però no gaire pràctic!

Estructures de Control i Python

Podem usar un estructura nova: un cicle indefinit o condicional, que s'executa fins que no es dóna una condició.

```
while <condition>:  
    <body>
```

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



Estructures de Control i Python

Iteracions interactives, o com repetir parts del codi sota demanda de l'usuari.

```
# average2.py

def main():
    sum = 0.0
    count = 0
    moredata = "yes"
    while moredata[0] == "y":
        x = eval(input("Enter a number >> "))
        sum = sum + x
        count = count + 1
        moredata = input("Do you have more numbers (yes or no)?")
    print("The average of the numbers is", sum / count)
```


Estructures de Control i Python

Iteracions sentinelles, o com repetir parts del codi fins que es compleix un cert valor.

```
# average3.py

def main():
    sum = 0.0
    count = 0
    x = eval(input("Enter a number (negative to quit) >> "))
    while x >= 0:
        sum = sum + x
        count = count + 1
        x = eval(input("Enter a number (negative to quit) >> "))
    print("The average of the numbers is", sum / count)
```

Estructures de Control i Python

Iteracions sentinelles (ii), o com repetir parts del codi fins que es compleix un cert valor.

```
# average4.py

def main():
    sum = 0.0
    count = 0
    xStr = input("Enter a number (<Enter> to quit) >> ")
    while xStr != "":
        x = eval(xStr)
        sum = sum + x
        count = count + 1
        xStr = input("Enter a number (<Enter> to quit) >> ")
    print("The average of the numbers is", sum / count)
```

Estructures de Control i Python

Suposem que volem mirar si dos punts estan en les mateixes coordenades:

```
if p1.getX() == p2.getX():
    if p1.getY() == p2.getY():
        # points are the same
    else:
        # points are different
else:
    # points are different
```

Això es pot fer molt més elegant amb expressions booleanes.

Una expressió **Booleana** és qualsevol expressió que avalua en dos possibles valors (0/1, veritat/fals).

Python proporciona tres operadors booleans: `and`, `or` i `not`.

Estructures de Control i Python

`<expr> and <expr>`
`<expr> or <expr>`

P	Q	$P \text{ and } Q$
T	T	T
T	F	F
F	T	F
F	F	F

P	Q	$P \text{ or } Q$
T	T	T
T	F	T
F	T	T
F	F	F

P	$\text{not } P$
T	F
F	T

L'ordre de precedència de Python és, primer NOT, seguit per AND i finalment OR.

`a or not b and c`

`(a or ((not b) and c))`

Estructures de Control i Python

Les operacions booleanes segueixen unes regles algebraiques molt concretes:

Algebra	Boolean algebra
$a * 0 = 0$	$a \text{ and } \text{false} == \text{false}$
$a * 1 = a$	$a \text{ and } \text{true} == a$
$a + 0 = a$	$a \text{ or } \text{false} == a$

$a \text{ or } \text{true} == \text{true}$

$a \text{ or } (b \text{ and } c) == (a \text{ or } b) \text{ and } (a \text{ or } c)$

$a \text{ and } (b \text{ or } c) == (a \text{ and } b) \text{ or } (a \text{ and } c)$

$\text{not}(\text{not } a) == a$

Llei de Morgan

$\text{not}(a \text{ or } b) == (\text{not } a) \text{ and } (\text{not } b)$

$\text{not}(a \text{ and } b) == (\text{not } a) \text{ or } (\text{not } b)$