UNIVERSITAT DE BARCELONA

Grau d'Enginyeria Informàtica
Facultat de Matemàtiques

Algorísmica I

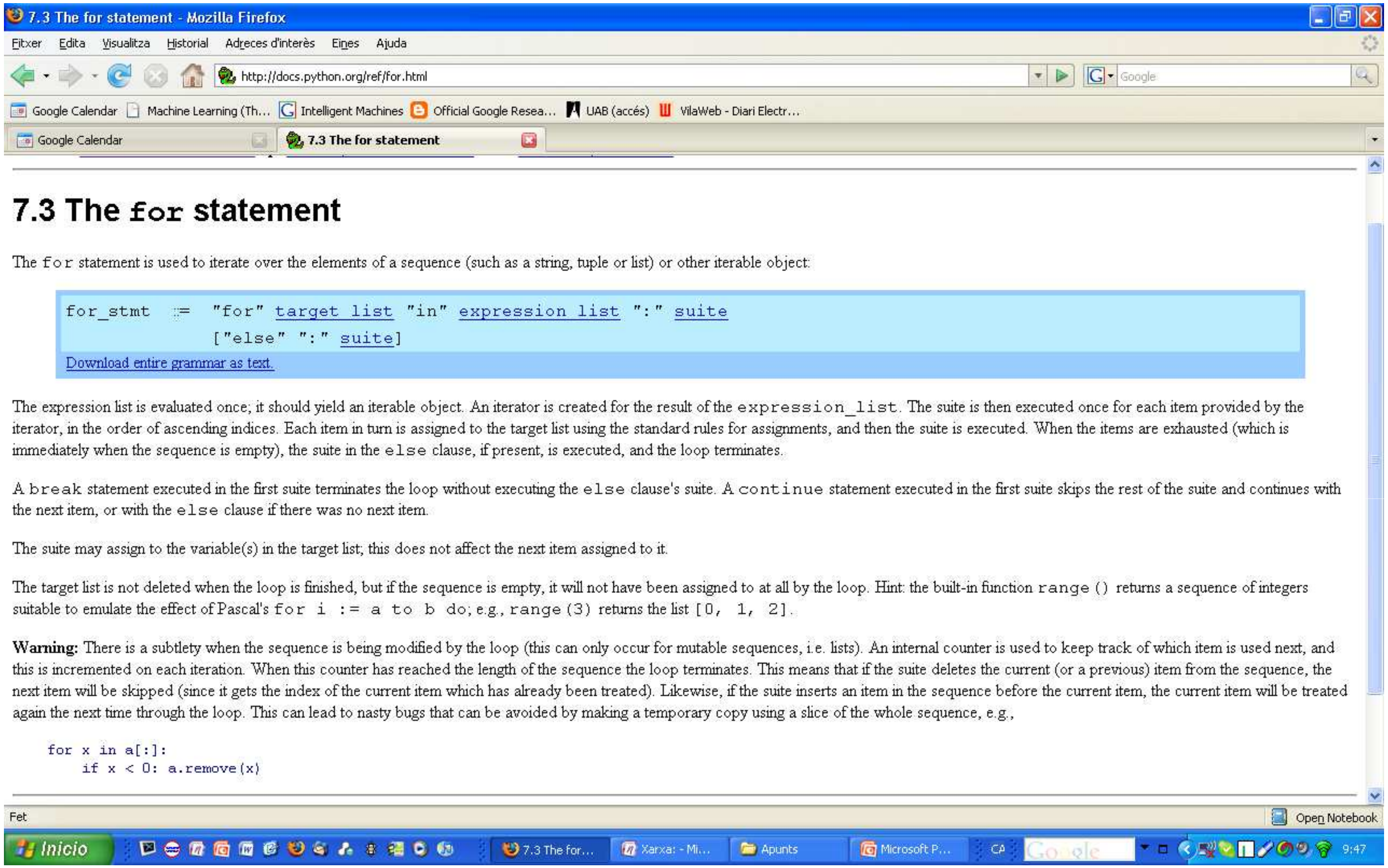
Introducció als algorismes II

Jordi Vitrià

Si teniu dubtes de programació en Python (què fa exactament una instrucció), podeu consultar a:

http://docs.python.org/ref

http://docs.python.org/contents.html


7.3 The for statement - Mozilla Firefox
Fitxer Edita Visualitza Historial Adreces d'interès Eines Ajuda
http://docs.python.org/ref/for.html
Google Calendar Machine Learning (Th... Intelligent Machines Official Google Resea... UAB (accés) VilaWeb - Diari Electr...
Google Calendar
7.3 The for statement
Fet
Open Notebook
Inicio
7.3 The for... Xarxa: - Mi... Apunts Microsoft P...
CA
9:47

7.3 The `for` statement

The `for` statement is used to iterate over the elements of a sequence (such as a string, tuple or list) or other iterable object:

```
for_stmt  ::=  "for" target_list "in" expression_list ":" suite
               ["else" ":" suite]
```

Download entire grammar as text.

The expression list is evaluated once; it should yield an iterable object. An iterator is created for the result of the `expression_list`. The suite is then executed once for each item provided by the iterator, in the order of ascending indices. Each item in turn is assigned to the target list using the standard rules for assignments, and then the suite is executed. When the items are exhausted (which is immediately when the sequence is empty), the suite in the `else` clause, if present, is executed, and the loop terminates.

A `break` statement executed in the first suite terminates the loop without executing the `else` clause's suite. A `continue` statement executed in the first suite skips the rest of the suite and continues with the next item, or with the `else` clause if there was no next item.

The suite may assign to the variable(s) in the target list; this does not affect the next item assigned to it.

The target list is not deleted when the loop is finished, but if the sequence is empty, it will not have been assigned to at all by the loop. Hint: the built-in function `range()` returns a sequence of integers suitable to emulate the effect of Pascal's `for i := a to b do`; e.g., `range(3)` returns the list `[0, 1, 2]`.

Warning: There is a subtlety when the sequence is being modified by the loop (this can only occur for mutable sequences, i.e. lists). An internal counter is used to keep track of which item is used next, and this is incremented on each iteration. When this counter has reached the length of the sequence the loop terminates. This means that if the suite deletes the current (or a previous) item from the sequence, the next item will be skipped (since it gets the index of the current item which has already been treated). Likewise, if the suite inserts an item in the sequence before the current item, the current item will be treated again the next time through the loop. This can lead to nasty bugs that can be avoided by making a temporary copy using a slice of the whole sequence, e.g.,

```
for x in a[:]:
    if x < 0: a.remove(x)
```

Strings i Python

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

```
>>> str1 = "Hello"
>>> str2 = ’spam’
>>> print str1, str2
Hello spam
>>> type(str1)
<type ’string’>
>>> type(str2)
<type ’string’>
```

Podem entrar *strings* des del teclat, amb compte!

```
>>> firstName = input("Please enter your name: ")
Please enter your name: John
Traceback (innermost last):
  File "<pyshell#8>", line 1, in ?
    firstName = input("Please enter your name: ")
  File "<string>", line 0, in ?
NameError: John
```

Strings i Python

Ho podem resoldre així:

```
>>> firstName = input("Please enter your name: ")
Please enter your name: "John"
>>> print "Hello", firstName
Hello John
```

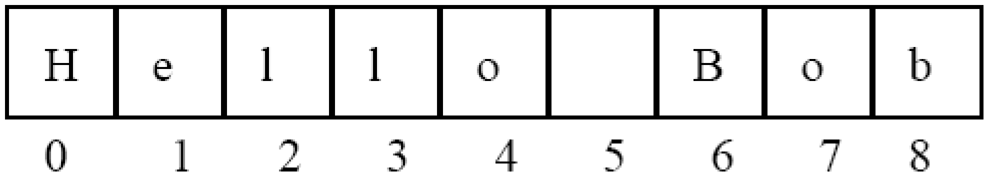
O així:

```
>>> firstName = raw_input("Please enter your name: ")
Please enter your name: John
>>> print "Hello", firstName
Hello John
```

Que és el mateix que `input()`, però sense avaluar l’expressió que entrem.

Strings i Python

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



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

```
>>> greet = "Hello Bob"
>>> greet[0]
'H'
>>> print greet[0], greet[2], greet[4]
H l o
>>> x = 8
>>> print greet[x-2]
B
```

```
>>> greet[0:3]
'Hel'
>>> greet[5:9]
' Bob'
>>> greet[:5]
'Hello'
>>> greet[5:]
' Bob'
>>> greet[:]
'Hello Bob'
```

Strings i Python

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

```
>>> "spam" + "eggs"
'spameggs'
>>> "Spam" + "And" + "Eggs"
'SpamAndEggs'
>>> 3 * "spam"
'spamspamspam'
>>> "spam" * 5
'spamspamspamspamspam'
>>> (3 * "spam") + ("eggs" * 5)
'spamspamspameggseggseggseggseggs'
>>> len("spam")
4
>>> len("SpamAndEggs")
11
>>>
```

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

Table 4.1: Python string operations

Strings i Python

Què fa aquest programa?

```
def main():



    # get user’s first and last names
    first = raw_input("Please enter your first name (all lowercase): ")
    last = raw_input("Please enter your last name (all lowercase): ")

    # concatenate first initial with 7 chars of the last name.
    uname = first[0] + last[:7]

    # output the username
    print "Your username is:", uname

main()
```

```
This program generates computer usernames.

Please enter your first name (all lowercase): elmer
Please enter your last name (all lowercase): thudpucker
Your username is: ethudpuc
```

Strings i Python

```python
# month.py
#  A program to print the abbreviation of a month, given its number

def main():
    # months is used as a lookup table
    months = "JanFebMarAprMayJunJulAugSepOctNovDec"

    n = input("Enter a month number (1-12): ")

    # compute starting position of month n in months
    pos = (n-1) * 3

    # Grab the appropriate slice from months
    monthAbbrev = months[pos:pos+3]

    # print the result
    print "The month abbreviation is", monthAbbrev + "."

main()
```

```
Enter a month number (1-12): 4
The month abbreviation is Apr.
```

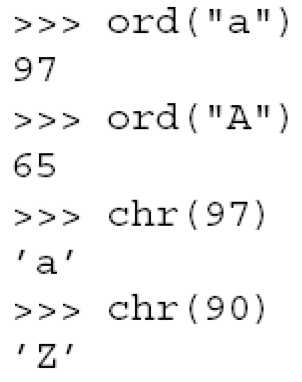
Strings i Python

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 els llengües. Usa 16 bits per caràcter. Per compatibilitat, és una superconjunt de l’ASCII.

Python ens dona funcions per accedir a aquests codis:



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

Source: www.LookupTables.com

Strings i Python

```
# text2numbers.py
#     A program to convert a textual message into a sequence of
#         numbers, utlilizing the underlying ASCII encoding.

def main():
    print "This program converts a textual message into a sequence"
    print "of numbers representing the ASCII encoding of the message."
    print

    # Get the message to encode
    message = raw_input("Please enter the message to encode: ")

    print
    print "Here are the ASCII codes:"

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

    print

main()
```

El `for` itera sobre strings, tuples o llistes

```
This program converts a textual message into a sequence
of numbers representing the ASCII encoding of the message.

Please enter the message to encode: What a Sourpuss!

Here are the ASCII codes:
87 104 97 116 32 97 32 83 111 117 114 112 117 115 115 33
```

Strings i Python

```
# numbers2text.py
#     A program to convert a sequence of ASCII numbers into
#         a string of text.

import string  # include string library for the split function.

def main():
    print "This program converts a sequence of ASCII numbers into"
    print "the string of text that it represents."
    print

    # Get the message to encode
    inString = raw_input("Please enter the ASCII-encoded message: ")

    # Loop through each substring and build ASCII message
    message = ""
    for numStr in string.split(inString):
        asciiNum = eval(numStr)            # convert digit string to a number
        message = message + chr(asciiNum) # append character to message

    print "The decoded message is:", message

main()
```

Strings i Python

```
>>> import string
>>> string.split("Hello string library!")
['Hello', 'string', 'library!']
```

Dividim strings en troços (blancs)

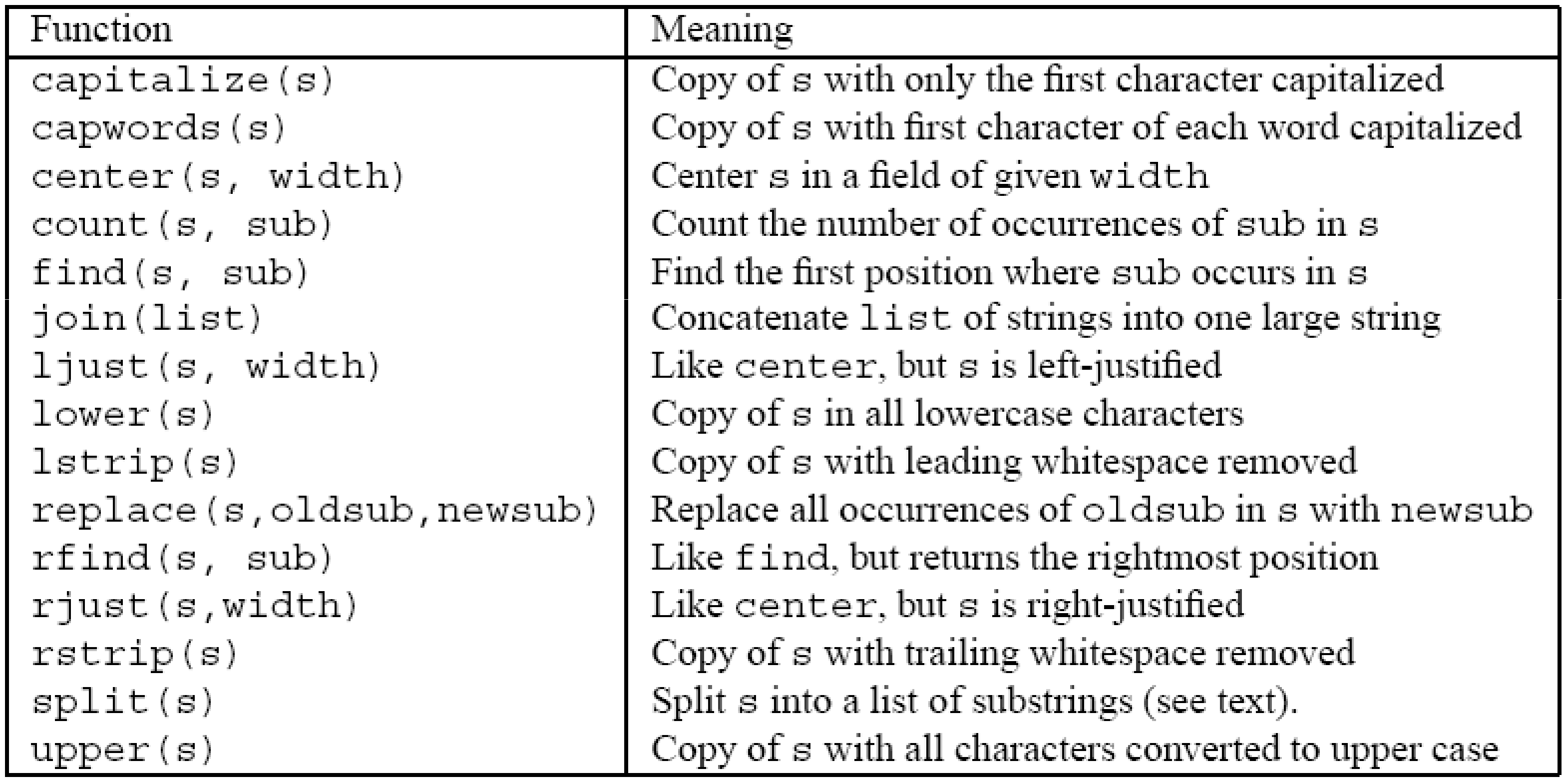
```
>>> string.split("32,24,25,57", ",")
['32', '24', '25', '57']
```

També podem especificar per “on” dividir!

```
>>> eval("345.67")
345.67
>>> eval("3+4")
7
>>> x = 3.5
>>> y = 4.7
>>> eval("x * y")
16.45
>>> x = eval(raw_input("Enter a number "))
Enter a number 3.14
>>> print x
3.14
```

```
>>> string.split("87 104 97 116 32 97 32 83 111 117 114 112 117 115 115 33")
['87', '104', '97', '116', '32', '97', '32', '83', '111', '117',
'114', '112', '117', '115', '115', '33']
```

Strings i Python



Function	Meaning
`capitalize(s)`	Copy of `s` with only the first character capitalized
`capwords(s)`	Copy of `s` with first character of each word capitalized
`center(s, width)`	Center `s` in a field of given `width`
`count(s, sub)`	Count the number of occurrences of `sub` in `s`
`find(s, sub)`	Find the first position where `sub` occurs in `s`
`join(list)`	Concatenate `list` of strings into one large string
`ljust(s, width)`	Like `center`, but `s` is left-justified
`lower(s)`	Copy of `s` in all lowercase characters
`lstrip(s)`	Copy of `s` with leading whitespace removed
`replace(s,oldsub,newsub)`	Replace all occurrences of `oldsub` in `s` with `newsub`
`rfind(s, sub)`	Like `find`, but returns the rightmost position
`rjust(s,width)`	Like `center`, but `s` is right-justified
`rstrip(s)`	Copy of `s` with trailing whitespace removed
`split(s)`	Split `s` into a list of substrings (see text).
`upper(s)`	Copy of `s` with all characters converted to upper case

Table 4.2: Some components of the Python string library

Strings i Python

Python també ens dona funcions per formatar *strings*:

```
<template-string> % (<values>)

%<width>.<precision><type-char>
```

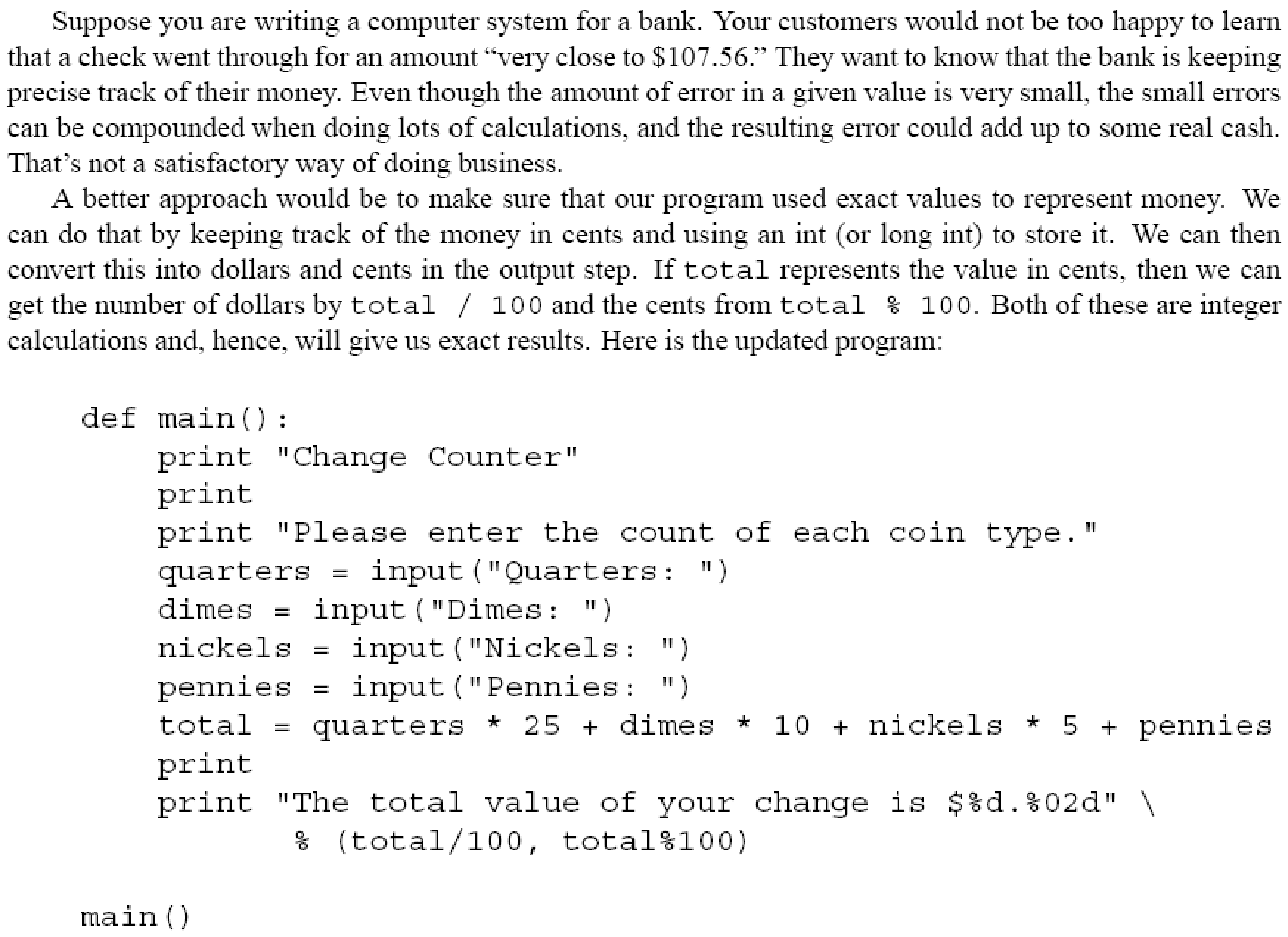
Operador de format

Especificador de format

decimal, float, string

```
>>> "Hello %s %s, you may have won $%d!" % ("Mr.", "Smith", 10000)
'Hello Mr. Smith, you may have already won $10000!'

>>> 'This int, %5d, was placed in a field of width 5' % (7)
'This int,     7, was placed in a field of width 5'

>>> 'This int, %10d, was placed in a field of width 10' % (7)
'This int,          7, was placed in a field of width 10'

>>> 'This float, %10.5f, has width 10 and precision 5.' % (3.1415926)
'This float,    3.14159, has width 10 and precision 5.'

>>> 'This float, %0.5f, has width 0 and precision 5.' % (3.1415926)
'This float, 3.14159, has width 0 and precision 5.'

>>> "Compare %f and %0.20f" % (3.14, 3.14)
'Compare 3.140000 and 3.14000000000000012434'
```

Strings i Python



Suppose you are writing a computer system for a bank. Your customers would not be too happy to learn that a check went through for an amount “very close to $107.56.” They want to know that the bank is keeping precise track of 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 (or long 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 us exact results. Here is the updated program:

```
def main():
    print "Change Counter"
    print
    print "Please enter the count of each coin type."
    quarters = input("Quarters: ")
    dimes = input("Dimes: ")
    nickels = input("Nickels: ")
    pennies = 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)

main()
```

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

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

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

Funcions i Python

O encara millor

```
# happy.py

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).

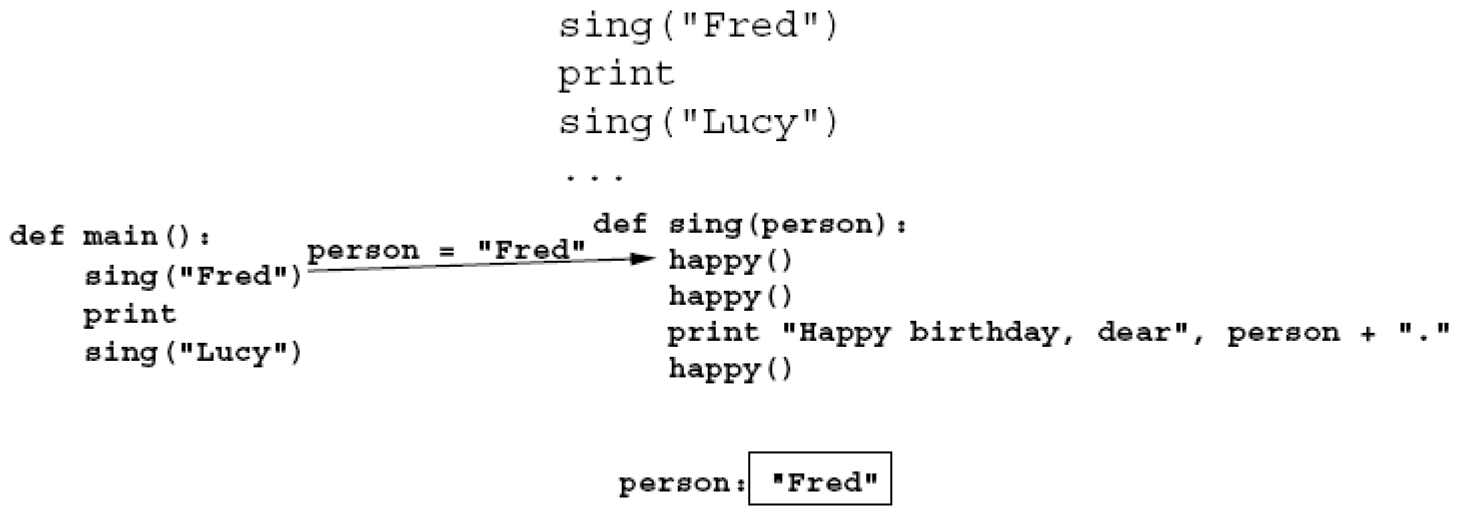
La funció es crida així:

```
<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.

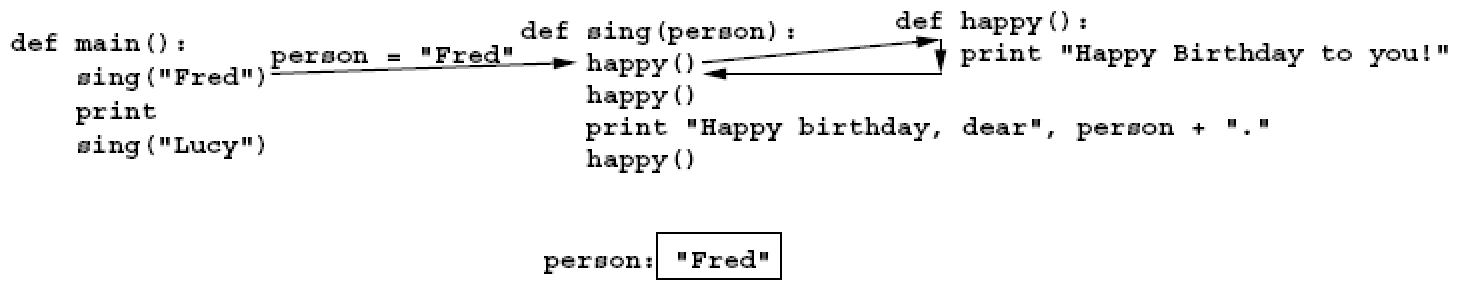


```
sing("Fred")
print
sing("Lucy")
...
```

```
def main():
    sing("Fred")
    print
    sing("Lucy")
```

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

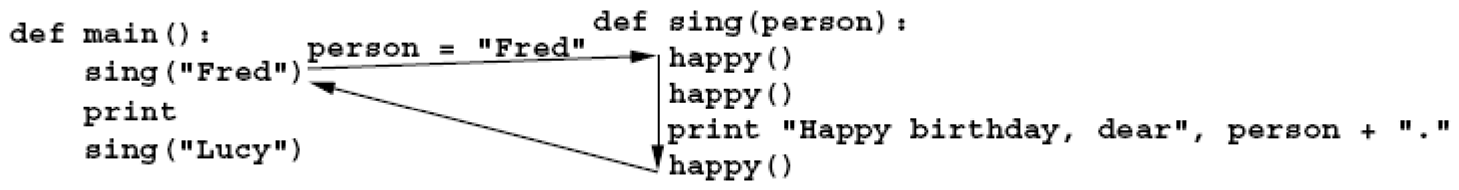
Funcions i Python



```
def main():
    sing("Fred")
    print
    sing("Lucy")
```

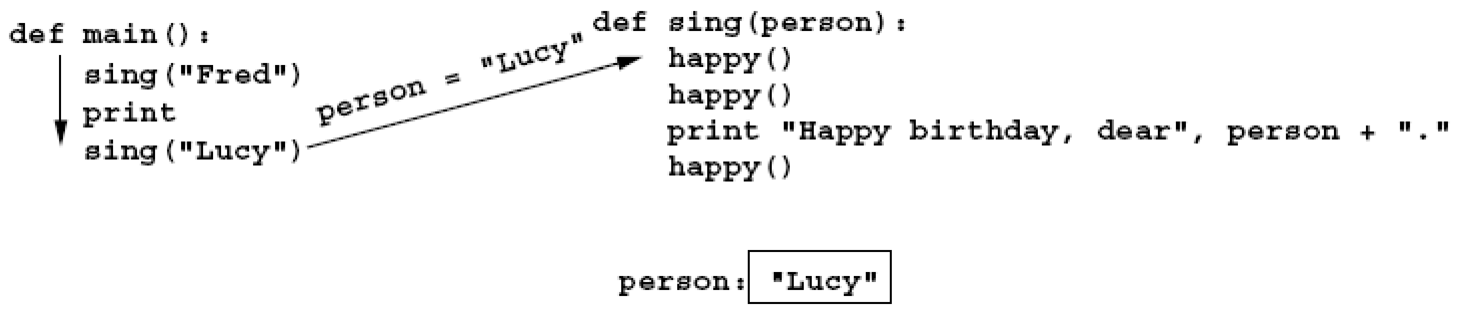
```
def sing(person):
    happy()
    happy()
    print "Happy birthday, dear", person + "."
    happy()
```

```
def happy():
    print "Happy Birthday to you!"
```



```
def main():
    sing("Fred")
    print
    sing("Lucy")
```

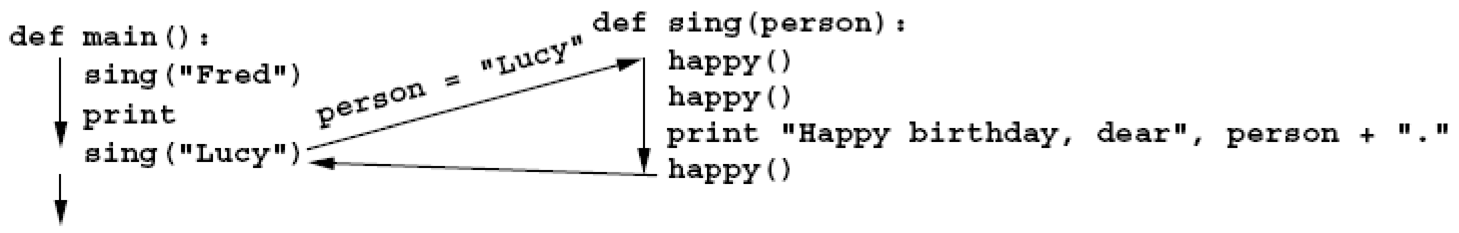
```
def sing(person):
    happy()
    happy()
    print "Happy birthday, dear", person + "."
    happy()
```



```
def main():
    sing("Fred")
    print
    sing("Lucy")
```

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

Funcions i Python



Funcions i Python

A vegades també volem que les funcions ens retornin valors:

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

Que s'escriuen així:

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

def distance(p1, p2):
    dist = math.sqrt(square(p2.getX() - p1.getX())
                     + square(p2.getY() - p1.getY())
    return dist


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

num1, num2 = input("Please enter two numbers (num1, num2) ")
s, d  = sumDiff(num1, num2)
print "The sum is", s, "and the difference is", d
```

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

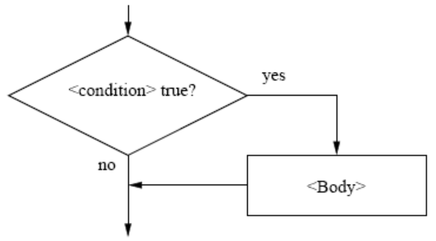
Fins ara, un programa és una seqüència pura d’instruccions, però a vegades necessitem trencar o alterar aquesta seqüència!

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 = 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 `int` (0 o 1).
Podem comparar nombres o *strings* (per ordre lexicogràfic).

Python	Mathematics	Meaning
<	$<$	Less than
<=	$\leq$	Less than or equal to
==	$=$	Equal to
>=	$\geq$	Greater than or equal to
>	$>$	Greater than
!=	$\neq$	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

Un cas important d'aplicació: al final de cada mòdul de Python hi podem posar:

```
if __name__ == '__main__':
    main()
```

La variable _name_ és instanciada automàticament pel Pyhton.
Si el mòdul és **importat**, li posa el nom del mòdul.

```
>>> import math
>>> math.__name__
'math'
```

Si és **cridat directament**, li dóna el valor __main__, i per tant és executat. Si és importat, no passarà res.

Estructures de Control i Python

Recordem:

```
# 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()
```

Aquest programa no funcionava en algunes ocasions...

Estructures de Control i Python

Podríem posar-hi:

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

dicrim = b * b - 4 * a * c
if discrim >= 0:
    discRoot = math.sqrt(discrim)
    root1 = (-b + discRoot) / (2 * a)
    root2 = (-b - discRoot) / (2 * a)
    print "\nThe solutions are:", root1, root2
```

EL seu funcionament seria:

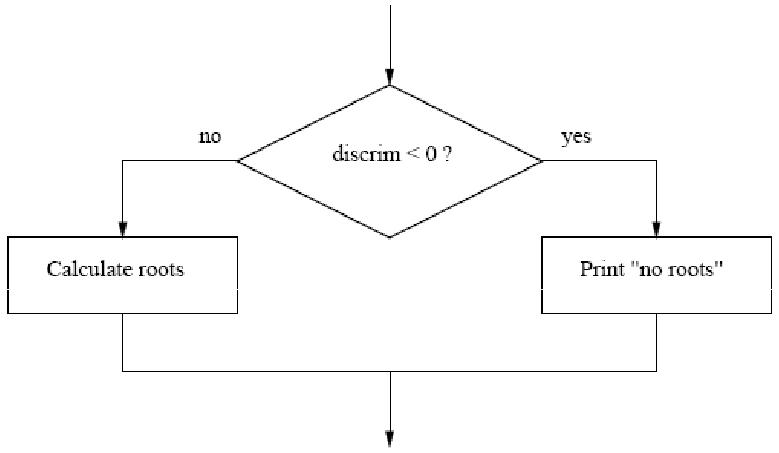
```
>>> quadratic2.main()
This program finds the real solutions to a quadratic

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:
        # Do stuff for two roots
```

O millor encara, així:

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

```
if discrim < 0:
    print "\nThe equation has no real roots!"
elif discrim == 0:
    root = -b / (2 * a)
    print "\nThere 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 "\nThe 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.

```
Input the count of the numbers, n
Initialize sum to 0
Loop n times
    Input a number, x
    Add x to sum
Output average as sum / n
```

```python
# average1.py

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

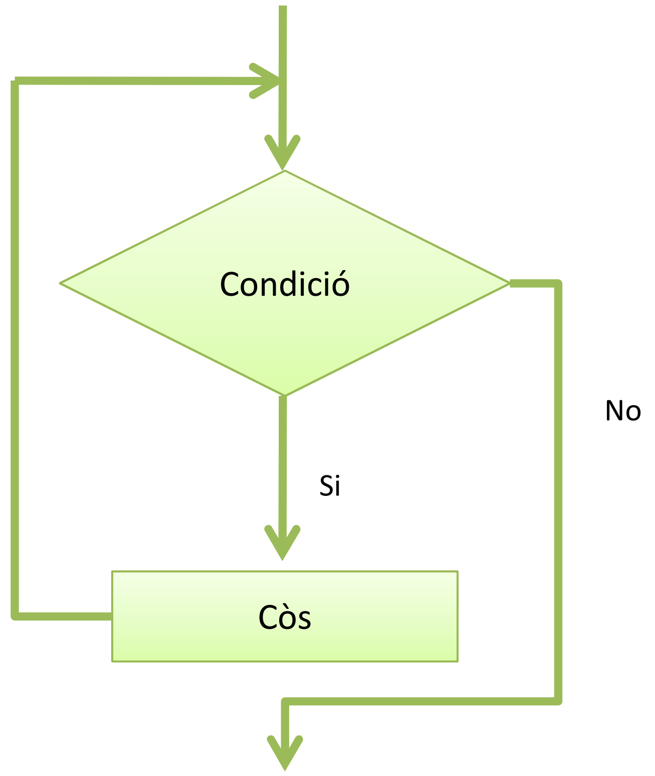
És correcte però no gaire pràctic!

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 = input("Enter a number >> ")
        sum = sum + x
        count = count + 1
        moredata = raw_input("Do you have more numbers (yes or no)? ")
    print "\nThe average of the numbers is", sum / count
```

Estructures de Control i Python

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

```
# average3.py

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

Estructures de Control i Python

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

```
# average4.py

def main():
    sum = 0.0
    count = 0
    xStr = raw_input("Enter a number (<Enter> to quit) >> ")
    while xStr != "":
        x = eval(xStr)
        sum = sum + x
        count = count + 1
        xStr = raw_input("Enter a number (<Enter> to quit) >> ")
    print "\nThe 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 and Q
T	T	T
T	F	F
F	T	F
F	F	F

P	Q	P or Q
T	T	T
T	F	T
F	T	T
F	F	F

P	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 and false == false
$a*1=a$	a and true == a
$a+0=a$	a or false == a

```
a or true == true
```

```
a or (b and c) == (a or b) and (a or c)
a and (b or c) == (a and b) or (a and c)
```

```
not(not a) == a
```

Llei de Morgan

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