Programació 1DAM/DAWsp

1r DAM - IES Jaume II el Just - Tavernes de la Valldigna

Abdó Garcia & Paco Galera

Continguts

1. UD1. Dades simples. Instruccions seqüencials

1.1 Introducció

Un programa és una seqüència d'instruccions que manipulen unes dades per a obtindre uns resultats.

Eixes instruccions són ordres que li fem a l'ordinador. Per a això cal dir-li-ho en el llenguatge que entén, que és el llenguatge màquina, compost per seqüències de 0s i 1s, igual que tota la informació que es guarda en un ordinador (números, text, fotos, música, jocs, pel·lícules...):

Però com per a nosaltres (els humans) ens resulta molt difícil, li ho direm en altre llenguatge. Començarem amb ***Python*** i més avant vorem ***Java***.

En este tema vorem les dades que són manipulades pels programes.

1.2 Dades simples

Una dada és qualsevol informació amb la qual treballa un algorisme.

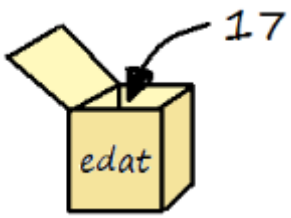
Cada dada és d'un tipus determinat que, bàsicament, serà enter, real, caràcter o lògic, però que dependrà del llenguatge de programació en què estem treballant.

Les dades apareixen en un programa en una de les següents formes:

- variables
- constants (simbòliques i literals)

Variables

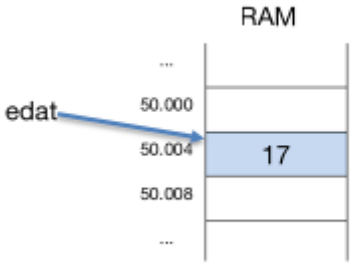
Una variable és un lloc on podem guardar una dada.



La imatge representa la variable **edat**, que guarda el valor **17**. Cada variable es caracteritza per tindre:

- Un **nom** (edat) i un **tipus** (enter), que s'han d'especificar quan es defineix la variable en un programa, amb una instrucció declarativa (encara que hi ha llenguatges, com *Python*, que no cal indicar el tipus).
- Un **valor** (17), que li s'assignarà en alguna instrucció d'assignació (o bé en la mateixa instrucció declarativa) i que podrà ser canviat per altre valor les voltes que calga.

Les variables s'emmagatzemen a la memòria RAM, de forma que:



- El **nom** (edat) representa l'adreça de la RAM on està el valor.
- El **tipus** (enter) especifica la quantitat de *bytes* necessaris per guardar un valor (4).
- El **valor** és el contingut (17).

```
// instrucció declarativa
int edat; // Definim una variable, de nom 'edat' i de tipus 'enter'

// instruccions d'assignació
edat = 17;          // Donem valor '17' a la variable edat
llig(edat);         // Assignem per teclat un valor a edat. Per exemple 19
edat = edat + 3;    // Tornem a canviar el valor. Ara valdrà 22
edat = 23.5;        // ERROR! Per què?

// utilització de la variable
escriu(edat);       // Escrivim en pantalla el valor de la variable edat
```

Nota

Més endavant veurem estes instruccions detalladament

Constants

Una constant és com una variable però que el valor no canvia durant l'execució del programa.

Les constants poden aparéixer en forma de literals o bé amb nom (constants simbòliques):

SIMBÒLIQUES	LITERALS
MAX_EDAT	99
PI	3.1416
VALOR_EURO	166.386
NOM_INSTITUT	"Jaume II el Just"
CICLES_INFORMATICA	true
MAJORIA_EDAT	18

Les constants simbòliques, igual que les variables, tenen un valor concret que se li dóna al principi del programa però, com ja hem dit, **no poden canviar de valor**.

Les constants literals alfanumèriques han d'expressar-se tancades entre cometes.

1.3 Expressions

Les constants i variables no apareixen aïllades, sinó formant part d'expressions. Una expressió és un càlcul necessari per a obtindre un resultat.

Una expressió és una combinació d'**operands** units mitjançant **operadors**.

- Els **operands** poden ser de diferents tipus:
- Literals: "Jaume II el Just", 100
- Constants: PI
- Variables: edat
- Funcions: arrel(100), longitut(nom)
- Els **operadors** els vorem en altre apartat

Exemples d'expressions

- Numèriques:
- edat
- 5
- 2 * PI * quadrat(radi)
- (-b+arrel(quadrat(b)-(4*a*c)))/(2*a)
- Alfanumèriques:
- "Neus"
- "Miquel" + "Garcia" + "Marqués"
- Lògiques:
- true
- false
- valor1 < valor2
- (valor1 < valor2)&&(valor2 <valor3)

1.4 Operadors

Són els símbols de les operacions amb els quals es construeixen les expressions.

Depenent del tipus de dades dels operands o del tipus del resultat, tenim uns **tipus** d'**operadors**: aritmètics, lògics, relacionals i alfanumèrics.

Operadors aritmètics

Són les operacions matemàtiques. Les variables o constants que hi intervenen són **numèriques** (enters o reals) i el resultat també. Els més usuals són:

OPERADOR	SIGNIFICAT
^	Exponenciació
*	Producte
/	Divisió
%	Residu de divisió entera
+	Suma
-	Resta

Regles de prioritat

Les expressions que tenen 2 o més operands necessiten unes regles que permeten determinar en quin ordre s'avaluen. Per exemple, si escrivim `escriu(2*5-3);` què mostrarà? 7 o 4? La resposta és 7, ja que les regles de prioritat indiquen que l'operador del producte té més prioritat que el de la resta, com veiem a la taula:

OPERADOR	PRIORITAT
^	ALTA
* / %	|
+ -	BAIXA

Si dos operadors d'igual prioritat coincideixen en una mateixa expressió, s'avaluen **d'esquerra a dreta**. Però si volguèrem canviar l'ordre d'avaluació en una expressió, utilitzarem els **parèntesi** necessaris. A banda, és recomanable l'ús de parèntesis davant del dubte.

Operadors relacionals

Serveixen per a comparar 2 expressions, retornant un valor lògic: vertader o fals.

OPERADOR	SIGNIFICAT
<	Menor
>	Major
==	Igual
!=	Distint
<=	Menor o igual
>=	Major o igual

Exemples d'expressions lògiques

Si **x** = **10** i **y** = **20**:

EXPRESSIÓ	VALOR
(x + y) < 20	false
(y - x) <= x	true
(y - x) >= x	true
x == y	false
x != y	true
'c' < 'f'	true *

* També podem comparar caràcters (van entre cometes)

Operadors lògics

Els operadors lògics són ***NO***, ***I*** *i* **O**. *Però per seguir la nomenclatura estàndard dels algorismes utilitzarem els noms anglesos:* **NOT**, **AND** *i* **OR**. *Actúen sobre operands o expressions lògiques i el resultat també és un valor lògic, que ve donat per les corresponents* **taules de veritat***, on V és Vertader (true) i F és Fals (false):

x	NOT x
F	V
V	F

x	y	x AND y
F	F	F
F	V	F
V	F	F
V	V	V

x	y	x OR y
F	F	F
F	V	V
V	F	V
V	V	V

Exemples

`NOT (3 < 5)` -> **F**

`(3 < 5) AND (4 < 2)` -> **F**

`(3 < 5) OR (4 < 2)` -> **V**

Expressions sinònimes

NOT(a < b)	a >= b	
NOT(a <= b)	a > b	
NOT(a + b == 0)	a + b != 0 Compte! No canvien els operandors aritmètics	
NOT(true)	false	
NOT(false)	true	
NOT(jubilat == true)	NOT(jubilat)	jubilat == false
NOT(jubilat == false)	jubiat == true	jubilat

Lleis de De Morgan

Són regles que permeten transformar expressions lògiques en altres equivalents. Són molt útils per a simplificar expressions condicionals. Concretament, transformen expressions formades amb un NOT sobre alguna expressió que té dins algun AND, OR o NOT.

1A LLEI DE DE MORGAN

`NOT(A AND B)` és equivalent a `NOT(A) OR NOT(B)`

Exemple

Si A és *Plou*, i B és *Fa fred*:

`NOT(A AND B)` significa *No és cert que ploga i faça fred alhora*. Això, segons esta llei, és equivalent a dir: `NOT(A) OR NOT(B)`, que significa *No plou o no fa fred*. Veiem-ho d'una altra forma, amb les variables *plou* i *fred*:

```
NOT(plou AND fred) -> NOT(plou) OR NOT(fred)
```

És a dir, tinguen els valors que tinguen *plou* i *fred* (verdader o fals), sempre tindrem el mateix resultat en les 2 expressions equivalents. Comprovem-ho amb la taula de veritat:

p(plou)	f(fred)	p AND f	NOT(p AND f)	NOT(p)	NOT(f)	N
V	V	V	**F**	F	F	
V	F	F	**V**	F	V	
F	V	F	**V**	V	F	
F	F	F	**V**	V	V	

2A LLEI DE DE MORGAN

`NOT(A OR B)` és equivalent a `NOT(A) AND NOT(B)`

Exemple

Si A és *Plou*, i B és *Fa fred*:

`NOT(A OR B)` significa *No és cert que: ploga o façca fred*. Això, segons esta llei, és equivalent a dir: `NOT(A) AND NOT(B)`, que significa *No plou i no fa fred*. Veiem-ho ara amb les variables *plou* i *fred*:

```
NOT(plou OR fred) -> NOT(plou) AND NOT(fred)
```

És a dir, tinguen els valors que tinguen *plou* i *fred* (verdader o fals), sempre tindrem el mateix resultat en les 2 expressions equivalents. Comprovem-ho amb la taula de veritat:

p(plou)	f(fred)	p OR f	NOT(p OR f)	NOT(p)	NOT(f)	NO
V	V	V	**F**	F	F	
V	F	V	**F**	F	V	
F	V	V	**F**	V	F	
F	F	F	**V**	V	V	

3A LLEI DE DE MORGAN

`NOT(NOT(A))` és equivalent a `A`

Exemple

Dir que *No és cert que no plou* és el mateix que dir que *plou*.

Com aplicar estes lleis quan el NOT actúa sobre més d'un operador lògic?

Per exemple:

`NOT(plou AND NOT(fred) AND sol AND humitat)`

Caldria:

- Llevar el NOT que abarca tota l'expressió.
- Canviar els AND per OR i al revés.
- Posar un NOT a cada part.

És a dir:

`(NOT(plou) OR NOT(NOT(fred)) OR NOT(sol) OR NOT(humanitat))`

I, aplicant la llei de la doble negació, quedaria:

`NOT(plou) OR fred OR NOT(sol) OR NOT(humitat)`

Compte! Si en l'expressió que abarca el NOT hi ha ORs i ANDs (les 2 coses alhora), cal anar en compte en transformació, ja que, com anem a vore a continuació, l'AND és més prioritari que l'OR. La solució seria, abans de fer la transformació, posar els parèntesis que calguen, i després ja fer la transformació conservant els parèntesis.

Per exemple, si tenim:

`NOT(plou AND fred OR sol)`

Primer posarem parèntesis per tindre una solució equivalent. Com l'AND és més prioritari que l'OR, posarem els paréntesi així:

`NOT((plou AND fred) OR sol)`

Ara ja apliquem les lleis de *De Morgan*, conservant els parèntesis de dins:

`(NOT(plou) OR NOT(fred)) AND NOT(sol)`

Regles de prioritat

Com els operadors lògics i relacionals poden formar expressions juntament amb els aritmètics, també necessitem unes **regles de prioritat** per a saber quins operadors s'avaluen primer.

OPERADOR	PRIORITAT
NOT	ALTA
^	|
*, /, %	|
+, -	|
<, >, <=, >=	|
==, !=	|
AND	V
OR	BAIXA

Estes regles són bastant estàndards però podria dependre de cada llenguatge de programació.

No obstant, davant el dubte, sempre podem (i devem) emprar els parèntesis.

1.5 Funcions

Són trossos de codi que podem utilitzar en els nostres programes. Hi ha de 2 tipus: predefinides i definides per l'usuari.

Funcions predefinides

Els llenguatges de programació tenen funcions predefinides amb les quals podem dur a terme les tasques més usuals. Les funcions (igual que en les de les matemàtiques) solen rebre un o més **arguments** i retornen un valor que anoomenem **resultat**.

Per exemple, per a mostrar coses per pantalla tenim:

- `printf("Hola");` en llenguatge C
- `System.out.println("Hola");` en llenguatge Java
- `escriu("Hola")` forma que emprarem en algorismes

Funcions definides per l'usuari

Podem crear funcions i usar-les en diferens parts del programa:

```
1       Programa principal{
2           escriu("L'àrea del  triangle de base 2 i altura 4 és:");
3           area = areaTriangle(2, 4);
4           escriu(area);
5           escriu("L'àrea del triangle de base 3 i altura 6 és:" + areaTriangle(3, 6));
6       }
7
8       // ------------------------------------------------
9
10      funció areaTriangle(base: enter, altura: enter){
11          area real;
12          area = base * altura / 2;
13          retorna area;
14      }
```

Ja vorem en detall l'ús de funcions més endavat...

1.6 Activitats

Activitat 1

Calcula el valor de cada expressió si és vàlida. Si no és vàlida, indica el motiu.

a. 10 * 3 + 5 * 2

b. 15 % 4

c. 2 + 7 / 3

d. 4 + "preu"

e. (5 + 2) < 8

f. 4 >= 4

g. true OR false

h. 5 OR (2 < 3)

i. (6 >= 2) OR (3 <= 5)

j. NOT(NOT(NOT(4 < 10>)))

k. 4 + false

l. 4 + 2 * 4 / 2

m. ((5 < 0) AND (6 >= 7)) OR (45 % 5 <= 0)

n. ((10 - 4) > 0) OR true

o. ((10 - 4) < 0) OR true

Activitat 2

Donats els següents valors de les variables **X = 1**, **Y = 4**, **Z = 10** i la constant **PI = 3.14**, avalua les expressions següents:

a. `2 * X + 0.5 * Y - 1 / 5 * Z`

b. `((PI * X ^ 2) > Y) OR ((2 * PI * X) <= Z)`

c. `"Hola, món!" == "Hola," + "món!"`

d. `'a' == 'A'`

Activitat 3

Construeix expressions correctes per a les fórmules següents:

$$ax^2 + bx + c \geq 0$$

$$\frac{3x - y}{z} - \frac{2xy^2}{z - 1} + xy$$

$$\frac{a}{b - \frac{c}{d - \frac{e}{f - g}}} + \frac{h + i}{j + k}$$

Activitat 4

A partir de les següents constants:

- **gran** = **fals**
- **redó** = **cert**
- **suau** = **fals**

...indica quin serà el valor de les següents expressions:

a. gran i redó i suau;

b. gran o redó o suau;

c. gran i redó o suau;

d. gran o redó i suau;

e. gran i (redó o suau);

f. (gran o redó) i suau;

Activitat 5

Indica amb parèntesis l'ordre en què l'ordinador executaria les diferents operacions:

a. x + y + z

b. x * y + z

c. x + y * z

d. x - y * z

e. x + y / z

f. x * y / z

g. x / y / z

h. x / y * y + x % y

i. x / y + z + x

Activitat 6

Tranforma les següents expressions en altres equivalents utilitzant les lleis de *De Morgan*. Cal tindre en compte que **a**, **b**, **c** són variables enteres i **p**, **q**, **r** són variables booleanes (lògiques).

a. `NOT((p AND q) OR r)`

b. `NOT((a == b) OR (a == 0))`

c. `NOT(NOT p OR NOT q OR (a == b + c))`

d. `NOT(p AND (q OR r))`

e. `NOT((a < b) AND (b < c))`

f. `NOT(NOT p AND q OR NOT r)`

g. `NOT(NOT(a != b) OR (a + b == 7))`

h. `NOT((a / b == 0) OR (a == c))`

Activitat 7

Donats els valors inicials de les següents variables enteres i lògiques:

```
a = 3; b = 5; c = 7; p = cert; q = fals;
```

... indica els valors que tindran estes variables després de les següents assignacions.

NOTA: en cada apartat es tindrà en compte els canvis de les variables dels apartats anteriors.

a. `a = 3 * b`

b. `b = a + c`

c. `p = p and (c > b)`

d. `q = p or q`

e. `r = a == b`

f. `a = a + 1`

g. `b = b - 2`

h. `a = a`

i. `b = b / 2 + c % 3`

Activitat 8

Sent **a**, **b**, **c** i **d** variables numèriques, escriu l'expressió lògica corresponent a:

a. Els valors de b i c són tots dos superiors al valor de d

b. a, b i c són idèntics

c. a, b i c són idèntics però diferents de d

d. b està comprés, estrictament, entre els valors de a i c

e. b està comrprés, estrictament, entre els valors de a i c, i el valor de a és més xicotet que el valor de c

f. Hi ha, com a mínim, dos valors idèntics entre a, b i c

g. Hi ha dos valors idèntics entre a, b i c, i només dos

h. Hi ha, com a màxim, dos valors idèntics entre a, b i c

Activitat 9

Escriu l'expressió algorísmica de les següents expressions:

a. Avaluar si el contingut d'una variable numèrica és divisible per 10 o per 7

b. Avaluar si una variable *preu* no és menor de 100½ ni major de 200€

Activitat 10

Si DN, MN, AN representen el dia, mes i any d'una persona i DA, MA, AA el dia, mes i any actuals, expresssa amb una expressió si la persona ha complit 18 anys.

Activitat 11

En un algorisme que analitza els resultats d'exàmens, hi ha 5 variables definides:

```
char opcio;             // Tipus d'alumne: (C)iències o (L)letres
int nl, nv, nm, nf;     // Notes de literatura, valencià, mate i física d'un alumne
```

Totes les notes estan calculades sobre 10 i tenen el mateix pes per a fer la mitjana.

Escriu les expressions lògiques corresponents a:

a. La mitjana de les quatre notes és superior a 5

b. Les notes de mate i valencià són superiors a la mitjana de les quatre notes

c. Hi ha, com a mínim, una nota superior a 5

d. Totes les notes són superiors a 5

e. La mitjana de les quatre notes és superior o igual a 5, i la mitjana de les notes de l'opció que ha agafat l'alumne també

2. UD2. Programació estructurada. Disseny d'algorismes en Python

2.1 Concepte de programació estructurada

És unconjunt de tècniques de programació que incorporen:

- **Disseny descendent**: tècnica consistent en descompondre successivament accions complexes en accions més simples. Divideix i venceràs. Funcions.
- **Estructures de control**: descriuen el flux d'execució d'una successió d'accions:
- **Seqüencial**: s'executen les ordres de dalt cap a baix.
- **Bifurcació**: executar un o altre conjunt d'instruccions, depenent d'alguna condició.
- **Repetició (o bucle)**: repetir un conjunt d'instruccions mentre es complisca una condició.

2.2 Algorismes

Igual que abans de fer una casa convé primer fer els plànols, per a fer un programa (mitjanament llarg) cal descriure els passos a fer, independentment del llenguatge de programació que es vol utilitzar. Eixe conjunt de passos vindria a ser l'algorisme. Veiem una definició formal.

Què és un algorisme?

És una descripció clara i no ambigua de les accions necessàries per a solucionar un problema en un ordre determinat. És com un programa però no està escrit en cap llenguatge de programació en concret. Serveix per a indicar els distints passos que ha de tindre el programa sense entrar en detall.

Quins elements té un algorisme?

- **Instruccions**: d'entrada, d'eixida i d'assignació.
- **Estructures de control**: bifurcacions i repeticions.

Com es fa un algorisme?

Per expressar un algorisme es poden utilitzar diverses metodologies. Estes són les més comunes:

- **Llenguatge natural o informal**: com en una recepta de cuina.
- **Pseudocodi**: combinació entre llenguatge natural i llenguatge de programació.
- **Ordinograma (diagrama de flux)**: representació gràfica on hi ha uns símbols (accions) units per fletxes (que indiquen l'ordre d'execució).

Exemple

Volem fer un programa que calcule l'àrea d'un cercle a partir del radi que s'introduirà per teclat (si el radi no és negatiu).

Independentment del llenguatge que utilitzarem, podem descriure els passos que caldria fer. Veiem com seria l'algorisme en cadascuna de les 3 metodologies:

LLENGUATGE NATURAL O INFORMAL:

1. Demanar el radi per teclat.
2. Si el radi és positiu, calcular l'àrea i mostrar-la.
3. Si no, mostrar un missatge d'error.

PSEUDOCODI:

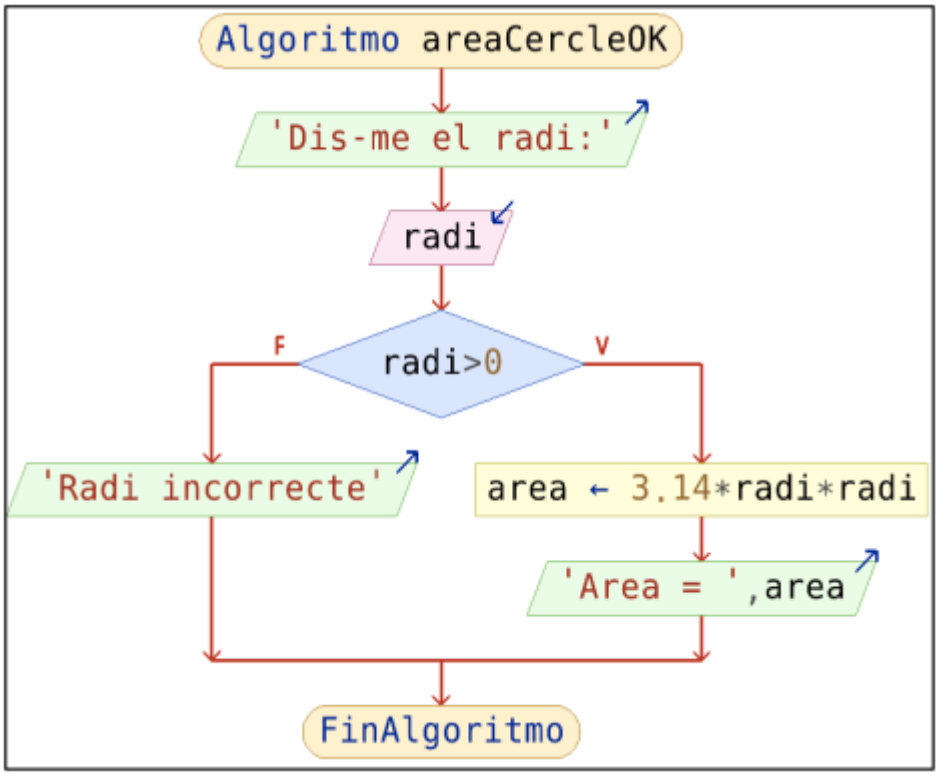
```
Algortimo areaCercleOK
    Escribir "Dis-me el radi:"
    Leer radi
    si radi > 0 Entonces
        area = 3.14 * radi * radi
        Escribir "Area = ", area
    Sino
        Escribir "Radi incorrecte"
    FinSi
FinAlgoritmo
```

Podem usar qualsevol altre nom per a les accions (en compte d'escribir, leer, etc...)

Però ací s'han usat eixos noms perquè és la sintaxi que usa l'aplicació *Pseint* (pseudo intèrpret d'algorismes), on podrem executar els algorismes (per si volguérem comprovar-ne el resultat).

PSEINT

ORDINOGRAMA (DIAGRAMA DE FLUX):



Amb *PSeInt* podem construir fàcilment este ordinograma a base d'anar posant eixes "caixetes". També podrem executar-lo.

PSeInt també permet convertir un ordinograma a pseudocodi (o un pseudocodi a ordinograma). I també podem exportar l'algorisme a un fitxer en *Python* a altre llenguatge.

Qualitat d'un algorisme

Per a resoldre un problema determinat hi pot haver infinitat d'algorismes. La qualitat d'un algorisme depén de:

- **Correctesa**: l'algorisme ha de produir el resultat correcte per a tots els *inputs* possibles.
- **Eficiència**: cal minimitzar els recursos de temps i memòria. Sobretot per a dades d'entrada "grans".
- **Senzillesa**: l'algorisme ha de ser el més senzill possible, sempre que siga fàcil d'entendre i modificar.

Estes qualitats ajuden a assegurar que un algorisme no només resol el problema plantejat, sinó que també ho fa de manera fiable, eficient i clara.

2.3 Elements d'un algorisme

Els elements d'un algorisme són un conjunt d'instruccions, de diferents tipus:

- **Instruccions d'entrada i d'eixida**: per a interaccionar amb l'usuari (introduir dades al programa o que aquest mostre resultats).
- **Instruccions d'assignació**: per a decidir si s'executen un conjunt d'instruccions o unes altres.
- **Instruccions de repetició**: per a repetir un conjunt d'instruccions.

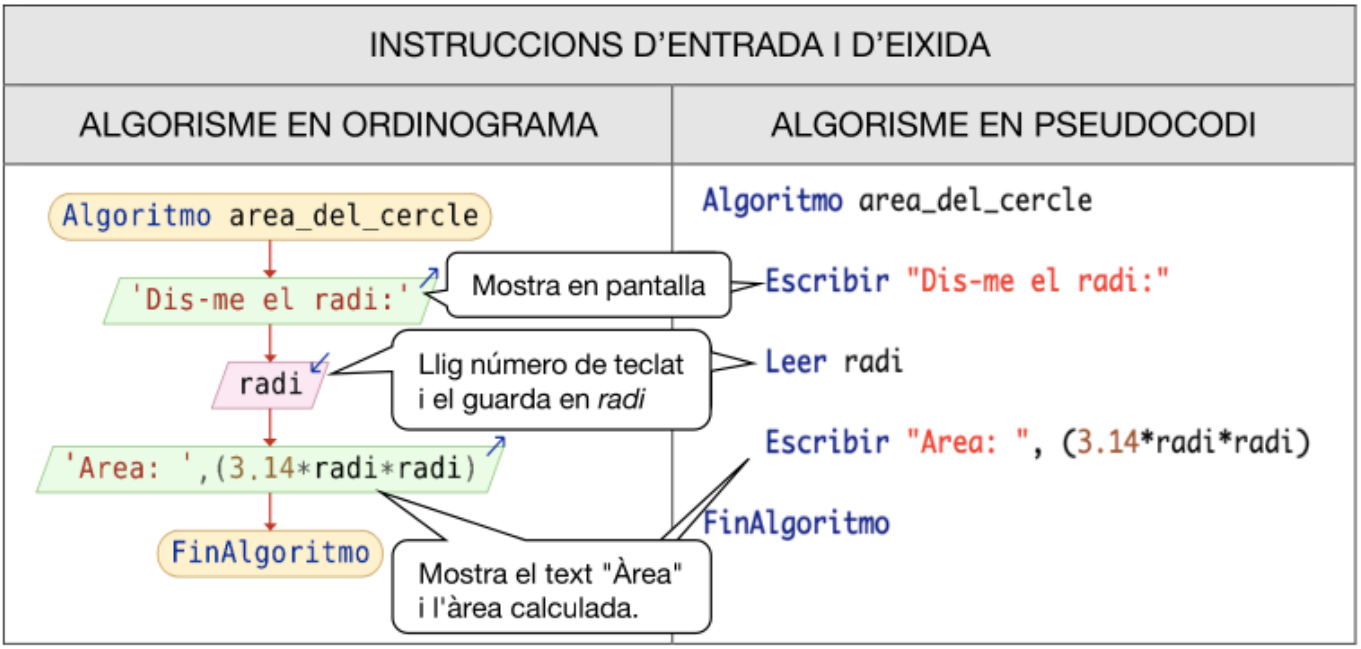
Anem a vore en detall en què consisteix cadascuna d'aquestes instruccions.

Instruccions d'entrada i d'eixida

Servixen perquè el programa intercanvie informació amb un medi extern (generalment teclat i pantalla).

- **Entrada (o lectura)**: es demana un valor per teclat (o pel ratolí, o des d'un fitxer o base de dades) per a guardar-lo en una variable (generalment).
- **Eixida (o escriptura)**: es mostra una dada (una constant o el valor d'una variable o el resultat d'una expressió) per pantalla (o per impressora o la deixarà en un fitxer o base de dades...).

Per a vore un exemple d'instruccions d'entrada i d'eixida, veiem com podria ser un algorisme que demane per teclat el radi d'una circumferència i mostre l'àrea corresponent.



Per iniciar-se en la programació, moltes vegades convé fer ús dels **ordinogrames** (sobretot quan vejam bifurcacions complexes, etc). Per tant, recomanem l'aplicació del *PSeInt* per tal de comprovar si l'algorisme funciona.

Ara bé, si volem fer l'algorisme en **pseudocodi**, quasi que és millor fer-ho ja en un llenguatge de programació que s'utilitze (no té sentit aprendre la sintaxi del *PSeInt*).

Per tant, anirem veient els distints elements en ordinograma i en *Python*.

Veiem com podria ser la implementació en *Python* d'este algorisme que hem vist:

```
print ("Dis-me el radi: ")  #(1)!
radi = int(input()) #(2)!
print("Àrea: ", 3.14 * radi * radi) #(3)!
```

1. Mostra la pantalla
2. Llig text, els converteix a enter i el guarda en la variable 'radi'
3. Mostra el text "Area" i el resultat del càlcul

Una altra forma:

```
import math #(1)!
radi = int(input("Dis-me el radi: ")) #(2)!
print("Àrea: ", math.pi * radi ** 2) #(3)!
```

1. Importem la llibreria per a usar un valor de pi més aproximat
2. Mostra text, llig valor per teclat, el converteix a enter i el guarda en 'radi'
3. Mostra el resultat. En Python la potència es fa amb **

Nota

• Per fer un ordinograma i poder-lo provar podem usar *PSeInt*:

• PSEINT

• Per a programar en local en *Python*:

• THONNY

• Es tracta d'un IDE molt senzill per a programar en *Python*. Ve integrat amb ell el propi *Python* i amb una única instal·lació podem començar a crear els nostres primers programes. **Recomanat!**

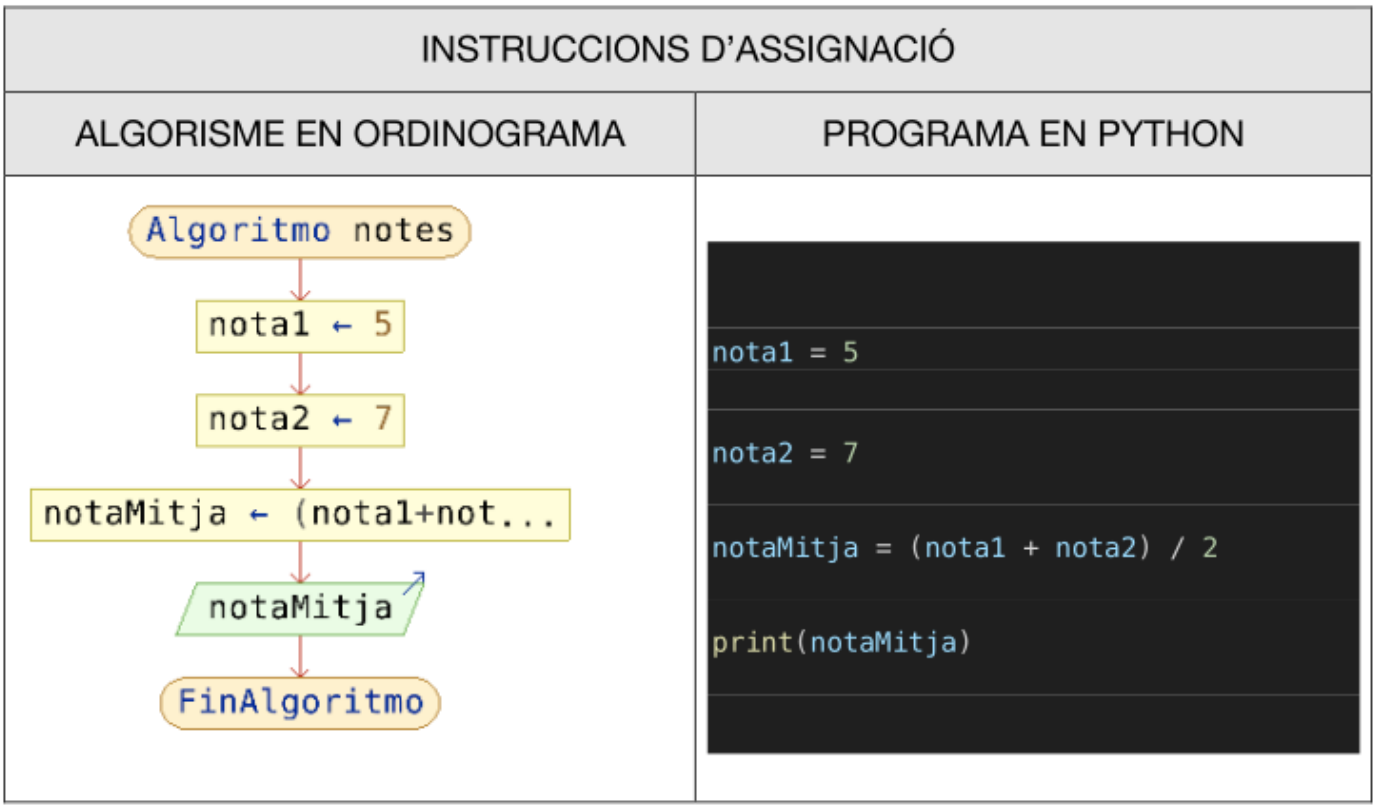
Al tema següent ja vorem altres alternatives com ***VSCode***.

Exercici 1. Instruccions d'entrada i d'eixida

Demana 2 números per teclat i mostra la seua suma, resta, multipliació i dividisió. Fes-ho de dues maneres: ordinograma (*PSeInt*) i en *Python* (OnlineGDB).

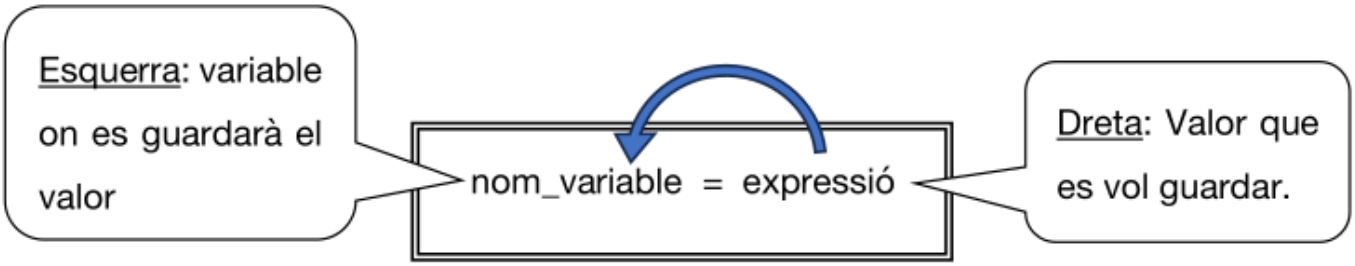
Instruccions d'assignació

Una assignació consisteix en guardar un valor en una variable.



En la primera instrucció de l'exemple anterior, està posant el valor 5 dins de la variable *nota1*. En la segona instrucció posa un 7 en la variable *nota2*. I en la tercera, suma les dos variables, les divideix entre 2 i el resultat el posa dins de la variable *notaMitja*.

És a dir, una assignació consta de 2 parts separades per un operador d'assignació. Sol emprar-se l'operador "=" (en *PSeInt* és una fletxa cap a l'esquerra: "<-"):



Cal tindre en compte que les assignacions NO són equacions matemàtiques.

Exemple

`x = x + 1` no t'e sentit com a equació però sí com a instrucció en un algorisme. És el que anomenem un ***increment de la variable***. En eixa instrucció se li assigna a la variable numèrica *x* un valor que correspon al valor que tenia abans eixa variable més una unitat.

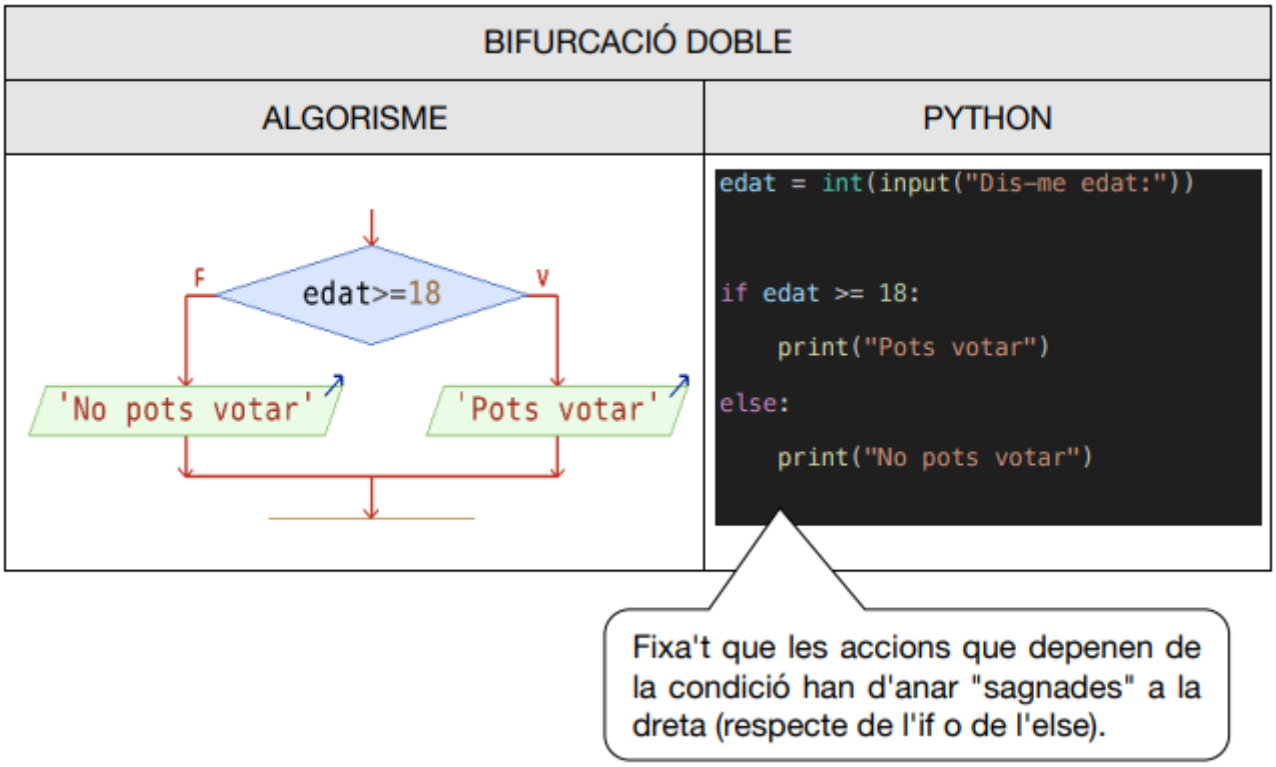
Exercici 2. Instruccions d'assignació

Fes l'algorisme amb un ordinograma o amb *Python* per a calcular el sou d'un treballador:

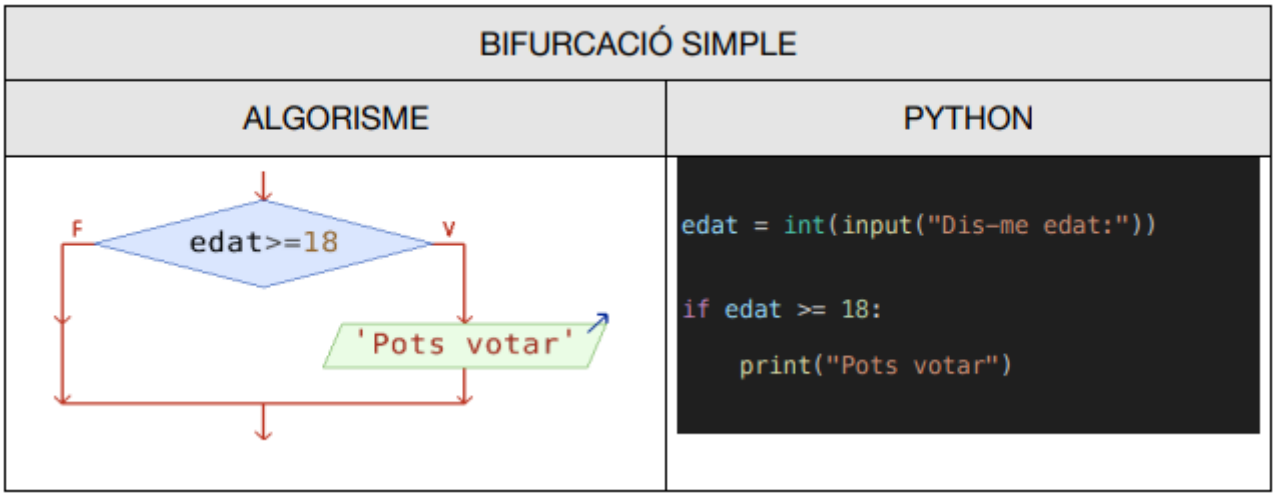
- Demana per teclat el nom del treballador, la quantitat d'hores que ha treballat i el preu per hora que paga l'empresa.
- Cal tenir en compte que la retenció aplicada és del 15%.
- Calcula el sou brut (import que paga l'empresa), l'import retingut (import que s'emporta hisenda) i el sou net (import que s'emporta el treballador).
- Mostra per pantalla el nom del treballador i les dades calculades abans.

Instruccions de bifurcació

Les instruccions de bifurcació (o selecció) serveixen per a quan volem executar un conjunt d'ordres **només** si es compleix alguna condició determinada.



En l'ordinograma podem no posar res en alguna de les 2 branques. En *Python* és opcional la part del *else*:



També podem posar instruccions de bifurcació dins d'altres. És a dir: una estructura *if* dins d'un altre *if*, o dins d'un altre *else*.

Exercici 3 RESOLT. Instruccions de bifurcació

Fes un programa en *Python* que calcule l'àrea d'un rectangle o que mostre un missatge d'error si algun costat no és positiu.

```
a = int(input("Llarg: "))
b = int(input("Ample: "))

if (a <= 0 or b <= 0):
    print("Costats incorrectes")
else:
    area = a * b
    print(f"L'àrea de {a} i {b} és {area}") # Hem fet ús dels "f-strings". Ho veurem més avant.
```

Exercici 4 RESOLT

Fes un algorisme que llisca 2 números i que mostre quin és el major; o bé, si és el cas, que diga que són iguals. Fixa't que hi ha un *if-else* dins d'un *else*.

```
n1 = int(input("Dis-me un número: "))
n2 = int(input("Dis-me'n un altre: "))

if (n1 > n2):
    print(f"El major és el {n1}")
else:
    if (n2 > n1):
        print(f"El major és el {n2}")
    else:
        print("Són iguals")
```

Exercici 5

Fes un programa en *Python* per a llegir un número de teclat i dir si és parell. No s'ha de dir res en cas contrari.

Exercici 6

Demanar un número i dir si és parell o imparell.

Exercici 7

Donats dos números, calcular quin és el més gran.

Exercici 8

Donats 3 números, calcular quin és el més gran.

Exercici 9

Donats 3 números, calcular quin és el més gran i el més menut.

Exercici 10

Donats 3 números, calcular quins són els dos més menuts.

Exercici 11

Donats 3 números, comprova si poden correspondre a les mesures dels costats d'un triangle.

PISTA: La suma dels dos més menuts ha de ser major que el gran.

Exercici 12

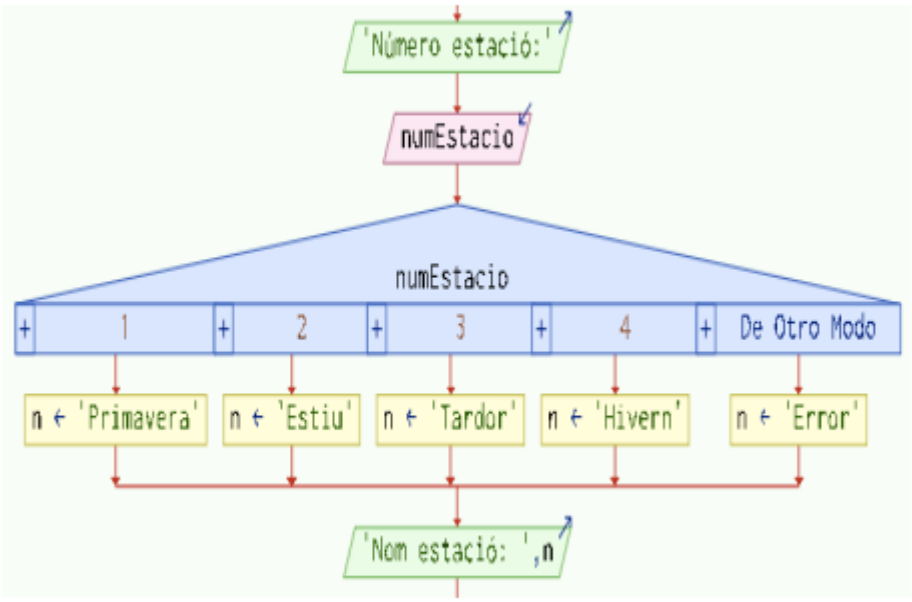
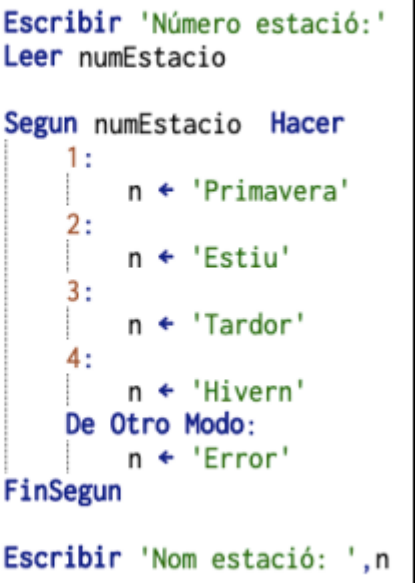
Transformar una nota numèrica a la forma: *Molt deficient, Insuficient, Suficient, Bé, Notable, Excel·lent*. També cal mostrar error si la nota és negativa o >10.

Exercici 13

Llegir dos números de teclat i una lletra, que serà el codi d'operació (Suma, Resta, Multiplicació o Divisió). Caldrà mostrar el resultat de l'operació demanada. Si no s'ha introduït un codi d'operació correcte, cal mostrar un error.

Instruccions de bifurcació múltiple

En la bifurcació doble, el programa executava un bloc d'instruccions d'entre 2 possibles, depenent del vaor d'una condició. Amb una instrucció de bifurcació múltiple, el programa executarà un bloc d'entre molts possibles, depenent del valor d'una variable (o expressió) entrea (o de tipus caràcter).

ORDINOGRAMA	PSEUDOCODI
	


```
Escribir 'Número estació:'
Leer numEstacio

Segun numEstacio  Hacer
    1:
        n ← 'Primavera'
    2:
        n ← 'Estiu'
    3:
        n ← 'Tardor'
    4:
        n ← 'Hivern'
    De Otro Modo:
        n ← 'Error'
FinSegun

Escribir 'Nom estació: ',n
```

A l'exemple, si el valor *numEstacio* és 1, s'executarà la instrucció (o conjunt d'instruccions) corresponent. Si fóra 2, les instruccions del 2, etc. I si no fóra ni 1, ni 2, ni 3 , ni 4, s'executaria el bloc d'instruccions de l'apartat *De Otro Modo*.

Ara bé: la bifurcació múltiple de *Python* no té eixa estructura. En compte de triar el bloc d'instruccions a executar segons el valor d'una sola variable (o expressió), en *Python* cada bloc d'instruccions té la seua condició per a poder executar-se.

Java sí que té una estructura semblant a la del *PSeInt* (anomenada ***switch***) però ja la veurem més endavant. Veiem ara com implementar la bifurcació múltiple en *Python*.

En *Python* s'utilitza l'estructura ***if-elif-else***, que és com si tinguérem dins d'un *else* una estructura *if-else*. I dins d'eixe *else*, una altra estructura *if-else*, etc.

BIFURCACIÓ MÚLTIPLE EN PYTHON

Diversos IF-ELSE niuats

```
numEstacio = int(input("Número estació:"))

if numEstacio == 1:
    no = "Primavera"
else:
    if numEstacio == 2:
        n = "Estiu"
    else:
        if numEstacio == 3:
            n = "Tardor"
        else:
            if numEstacio == 4:
                n = "Hivern"
            else:
                n = "Error"

print("Nom estació:", n)
```

Estructura IF-ELIF-ELSE

```
numEstacio = int(input("Número estació:"))

if numestacio==1:
    n = "Primavera"
elif numestacio==2:
    n = "Estiu"
elif numestacio==3:
    n = "Tardor"
elif numestacio==4:
    n = "Hivern"
else:
    n = "Error"



print("Nom estació: ",n)
```

Fixeu-vos que:

- Un *elif* ve a ser la contracció de *el(se)+if*, amb l'avantatge que no has d'anar sagnant cap a la dreta en cada condició (i no ocupa tantes línies).
- Esta estructura de *Pyhton* anirà per un camí o per altre depenent de les **diverses condicions** que li anem posant. És a dir: es podrien posar condicions diferents en cada *elif*. Mentre que en el *según* del *PSeInt*, anirà per un camí o per altre depenent dels valors d'una sola variable o expressió.

Exercici 14. Instruccions de bifurcació múltiple

Fes un programa en *Python* usant *elif* per a llegir de teclat un número de l'1 al 7 i mostrar el nom corresponent al dia de la setmana (dilluns, dimarts...):

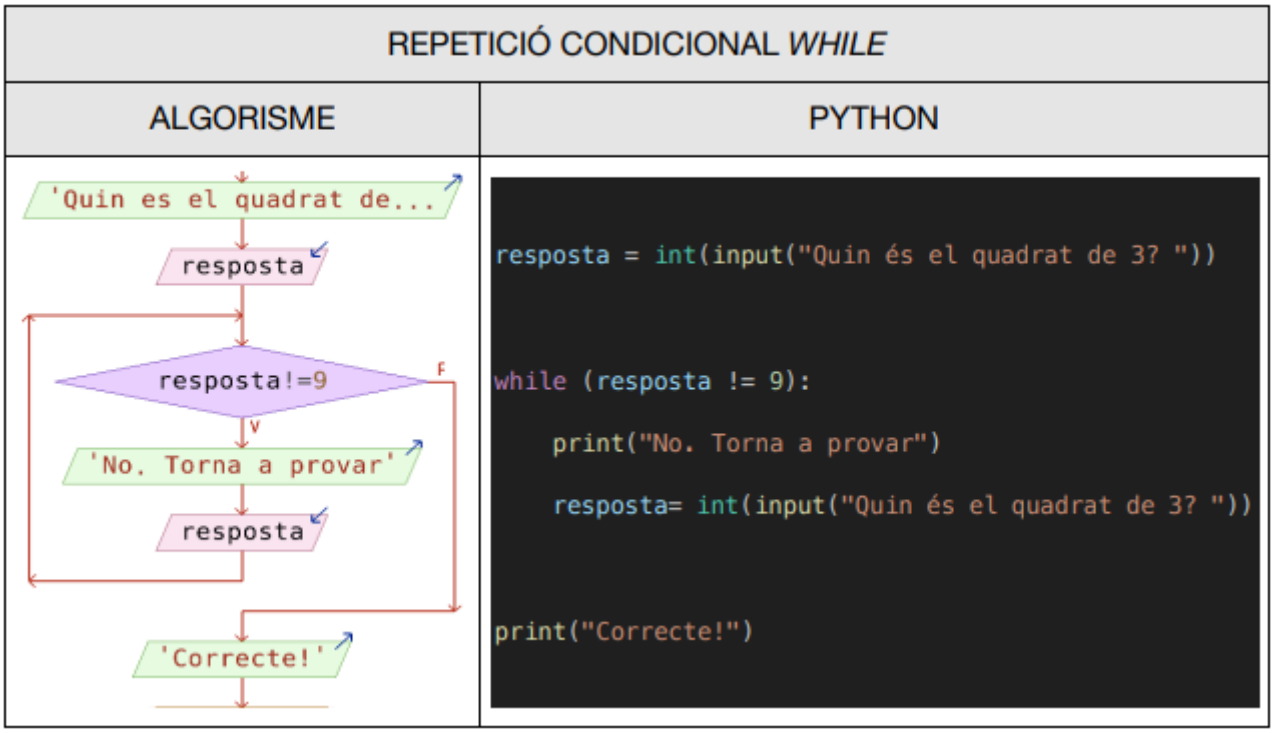
Instruccions de repetició

Amb les estructures de repetició podrem fer que un grup de sentències s'execute diverses vegades. Quantes?

- mentre es complisca una condició: **repeticions condicionals**.
- una determinada quantitat de vegades: **repeticions incondicionals**.

Repetició condicional WHILE

Amb l'estructura *while* posarem en un bloc aquelles instruccions que volem que s'executen repetidament **mentre** es complisca una determinada condició.



Exercici 15. Instruccions de repeticions condicionals *while*

Fes un programa en *Python* que, donat el radi, calcule l'àrea del cercle, però demanant repetidament el radi fins que l'usuari ens done un radi positiu.

Exercici 16

Demana any de naixement i de defunció d'una persona. Caldrà demanar-ho repetidament fins que siguen dades coherents (l'any de defunció no pot ser anterior al de naixement). Després, mostra quants anys ha viscut.

Repetició condicional *FINS* (no en *Python*)

Amb l'estructura algorísmica *FINS*, posarem en un bloc aquelles instruccions que volem que s'executen repetidament **fins que** es complisca una determinada condició.

És paregut al bucle *MENTRE (while)* però amb 2 diferències:

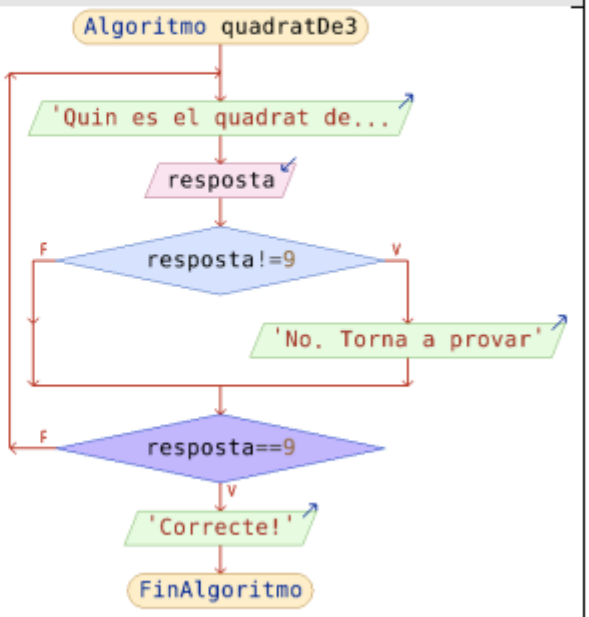
Exemple de repetició condicional *FINS*

REPETICIÓ CONDICIONAL *FINS*

PSEUDOCODI

```
Algoritmo quadratDe3

    Repetir

        Escribir "Quin es el quadrat de 3?"

        leer resposta

        si resposta != 9
            Escribir "No. Torna a provar"
        FinSi

    Hasta Que resposta == 9

    Escribir "Correcte!"

FinAlgoritmo
```

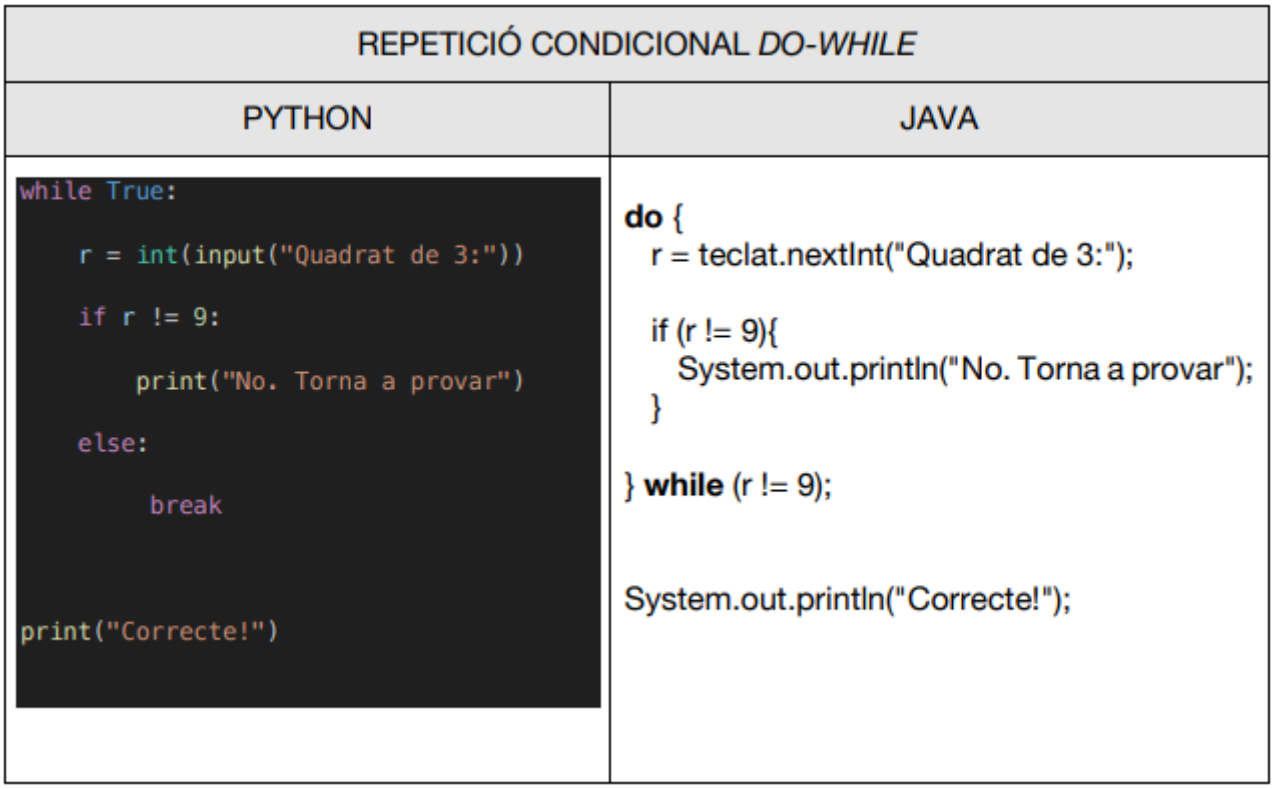
DIAGRAMA DE FLUX



Veiem que hi ha 2 diferències entre un *MENTRE* i un *FINS*. En un bucle *FINS*:

- **Sempre s'entra al bucle almenys 1 vegada**, ja que la condició està al final del bucle.
- **La condició és just la contrària** que *MENTRE*, ja que no hem de posar la condició per a continuar en el bucle, sinó la condició per a eixir d'ell. No és el mateix dir *"Mentre (no em donen la paga, la demanaré)"* que *"Repeteix demanar la paga fins que (sí que em donen la paga)"*.

Python no té esta estructura de posar la condició al final del bucle (no al principi), però podem simular eixe comportament (i *Java* té el ***do-while***, que és una cosa intermitja entre un *MENTRE* i un *FINS*):



En *Python* veiem que tenim un bucle infinit (en principi) ja que hem posat un *while true*. Però després, podem eixir del bucle ***a la força*** amb el *break*.

En canvi, en *Java* vorem que tenim l'estructura *do-while*, que és pareguda al *FINS* del *PSeInt* perquè la condició està al final però es diferencia en que la condició que posem és per a continuar en el bucle, no per a eixir.

Com sabem si cal usar la condició al principi o bé usar el *while true* i el *break*? Per regla general, si és una condició senzilla, cal posar la condició al principi. Si usem el *while true* i no posem els *break* que toquen, podriem provocar un **bucle infinit**.

Exercici 16 amb *while-true* i *break*

Demana any de naixement i de defunció d'una persona. Caldrà demanar-ho repetidament fins que siguen dades coherents (l'any de defunció no pot ser anterior al de naixement). Després, mostra quants anys ha viscut.

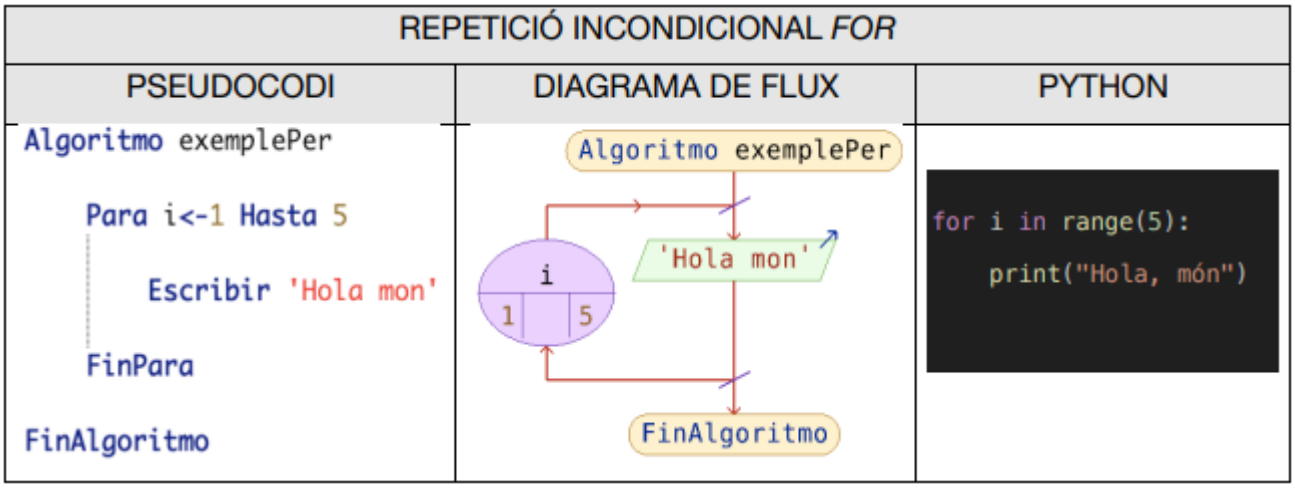
```
anyNaixement = int(input("Any de naixement: "))

while True:
    anyDefuncio = int(input("Any de defuncio: "))
    if anyDefuncio >= anyNaixement:
        break
    else:
        print("L'any de defuncio no pot ser anterior a l'any de naixement")

print("Ha viscut", anyDefuncio - anyNaixement, "anys")
```

Repetició incondicional FOR

Amb l'estructura *PER_A* (*para* en castellà i *for* als llenguatges de programació), podrem repetir un bloc **una quantitat de voltes determinada**. Per exemple, si volem que un bloc d'instruccions s'execute 5 vegades, ho farem en una estructura (*for*) amb ajuda d'una variable que anirà agafant els valors del 0 al 4 (o de l'1 al 5, etc).

Exemple

Volem mostrar 5 vegades *"Hola, món!"*



Exemple

Mostrar els números que hi ha entre el 100 i el 200, de 3 en 3.

Pseudocodi Python

```
Algoritmo AltreExemplePer
    Para i<-100 hasta 200 con paso 3
        Escribir i
    FinPara
FinAlgoritmo
```

```
for i in range(100, 201, 3):
    print(i)
```

Més endavant entrarem en detall en distints tipus de *for* que té *Python*. De moment, veiem alguns exemples dels valors que tindrà la ***i*** en funció del ***range***:

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

Exercici 17. Instruccions de repeticions incondicionals

Demanar per teclat quants números es volen mostrar. A continuació, es mostraran els números des d'eixe número fins a l'1 (en eixe ordre).

Exercici 18

Demanar un valor inicial i un valor final. Caldrà mostrar els valors que hi ha entre ells però de 3 en 3. El valor inicial pot ser major que el final.

Exemples:

- `vi = 10  vf = 20` ---> Mostrarà: 10, 13, 16, 19
- `vi = 20  vf = 10` ---> Mostrarà: 20, 17, 14, 11

Exercici 19

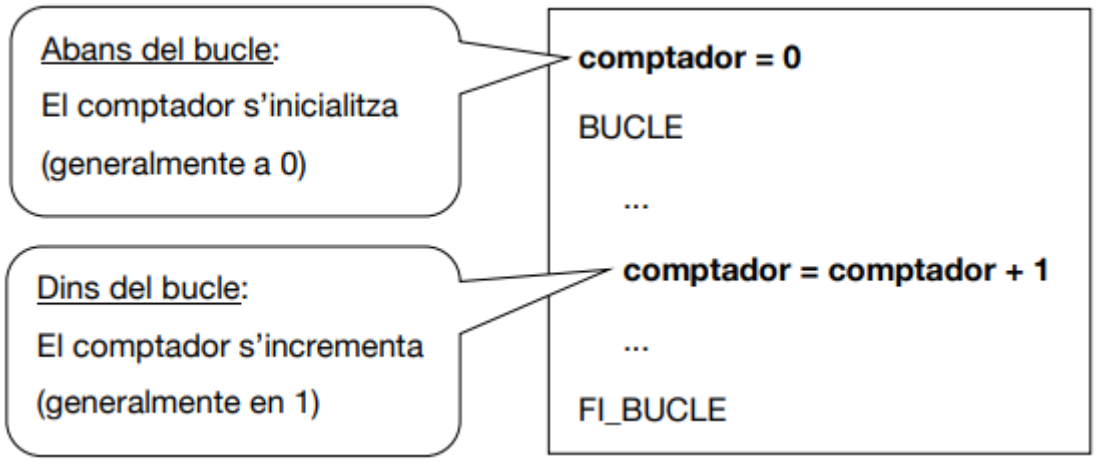
Imprimeix la taula de multiplicar del 9.

2.4 Comptadors, acumuladors i interruptors

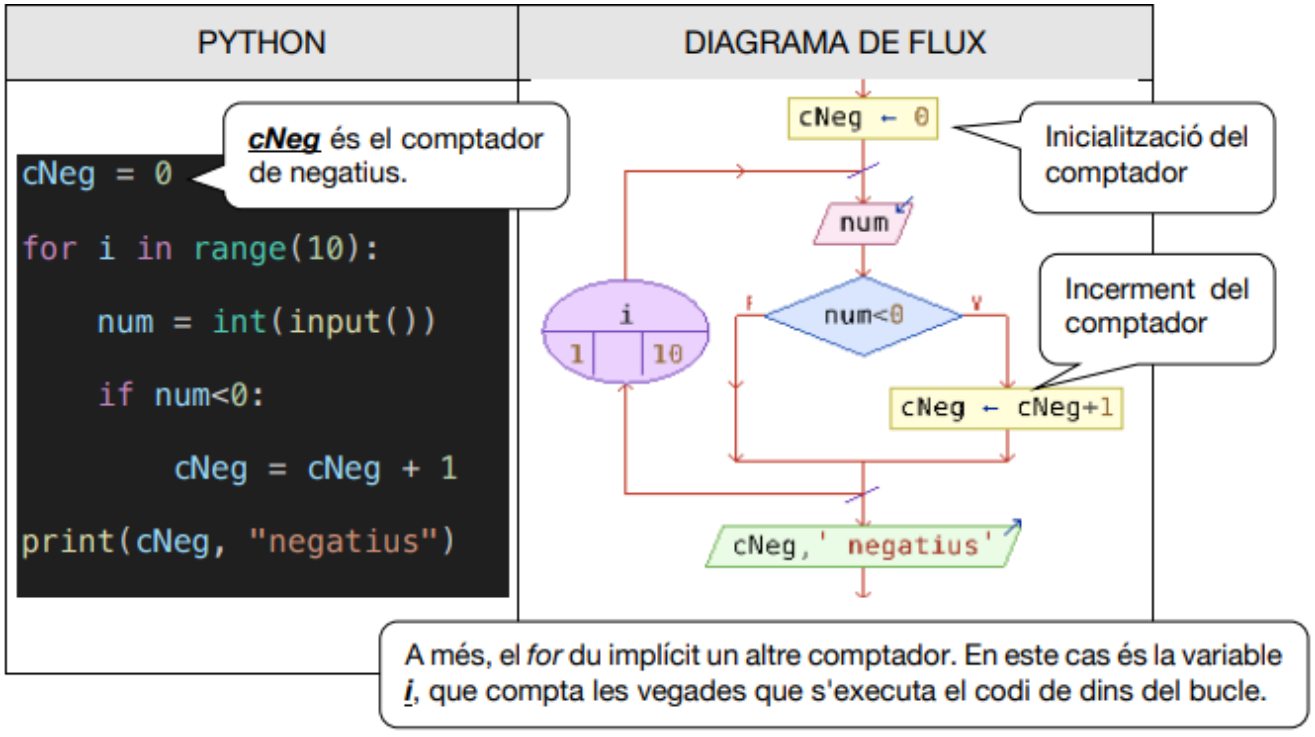
De vegades, les variable s'utilitzen per a unes finalitats concretes. Anem a vore quines són eixes finalitats i com tractar eixes variables.

Comptadors

Un comptador és una variable destinada a comptar quantes vegades ocorre alguna cosa. Sol emprar-se en els bucles (de qualsevol tipus: *while* o *for*). L'ús del comptador té 2 instruccions bàsiques i sol ser així:



Exemple d'ús de comptadors



Exercici 21. Ús de comptadors

Indica quants divisors (no quins) té un número donat.

Exercici 22

Pregunta quina és l'arrel quadrada de 225 fins que siga encertat. Finalment, mostra quants intents s'han fet.

Exercici 23

Imprimix quants números hi ha entre 1 i 100 que són múltiples de 2, quants múltiples de 3 i quants múltiples de 2 i de 3 al mateix temps.

Exercici 24

Llig uns quants números (fins que posem 0). Mostra quants positius, quants negatius i quants acaben en 0.

Exercici 25 RESOLT

Fes un programa que donat un número, calcule quants dígits té.

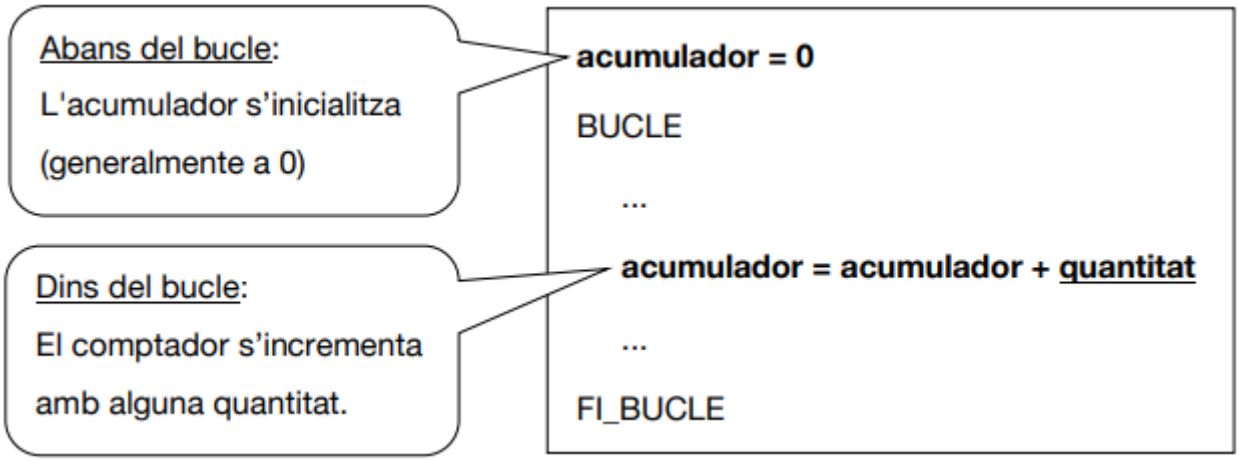
```
num = int(input("Dona'm un número "))

cDigits = 0 #(1)!

while (num > 0):
    num = num // 10 #(3)!
    cDigits = cDigits + 1 #(2)!

print(cDigits, "dígits")
```

1. inicialització del comptador
2. Increment
3. S'ha emprat l'operador // que torna sols la part sencera de la divisió

Acumuladors

Un acumulador és una variable destinada a acumular diferents quantitats.

Un acumulador és com un comptador però en compte de sumar 1, sumarem diferents quantitats (no volem *comptar* sinó *acumular* quantitats).



Exemple d'acumulador

Volem acumular l'import d'una factura:

```
q = int(input("Quantitat: "))

total = 0 #(1)!

while (q!=0):
    p = int(input("Preu: "))
    total = total + (q*p) #(2)!
    q = int(input("Quantitat: "))

print("Total:", total)
```

1. inicialització de l'acumulador
2. Increment

Exercici 26. Ús d'acumuladors

Demana quantitats fins que s'introduïsca la quantitat 0. Cadrà mostrar la suma de totes les quantitats.

Exercici 27

Demana les notes dels 23 alumnes de la classe. Mostra la nota mitjana.

Exercici 28

Mostra els números naturals que hi ha entre dos números introduïts per teclat i calcula la suma dels parells i la dels imparells. Per últim, mostra els dos totals.

Exercici 29

Introdueix 2 valors A i B (A < B). Incrementa A de 2 en 2 i decrementa B de 3 en 3 fins que A > B.

Exercici 30

Demana 2 números per teclat i mostra la multiplicació dels dos... però sense usar l'perador de la multiplicació (*). És a dir: hauràs de sumar un dels dos números tantes vegades com diu l'altre número.

Acumuladors de productes

Generalment, la quantitat va **sumant-se** a l'acumulador, però també podria **multiplicar-se**. Cal tindre en compte això per a iniciar l'acumulador:

- Si volem **sumar** quantitats, el `valor_inicial` sol ser **0**.
- Si volem **multiplicar** quanitats, el `valor_inicial` sol ser **1**.

Exercici 31 RESOLT

Donats 2 números (base i exp) calcula la potència ($base^{exp}$). Se suposa que la potència no és un operador ni cap funció predefinida.

```
base = int(input("Base: "))
expo = int(input("Exponent: "))

pot = 1 #(1)!

for i in range (expo):
    pot = pot * base #(2)!

print ("Potència:", pot)
```

1. Si inicialitzàrem a 0, el producte sempre donaria 0.
2. Acumulem el producte

Exercici 32. Acumuladors de productes

Mostra el producte de tots els números imparells entre l'1 i el 40.

Exercici 33

Mostra la suma, el producte i la mitjana dels 100 primers números naturals.

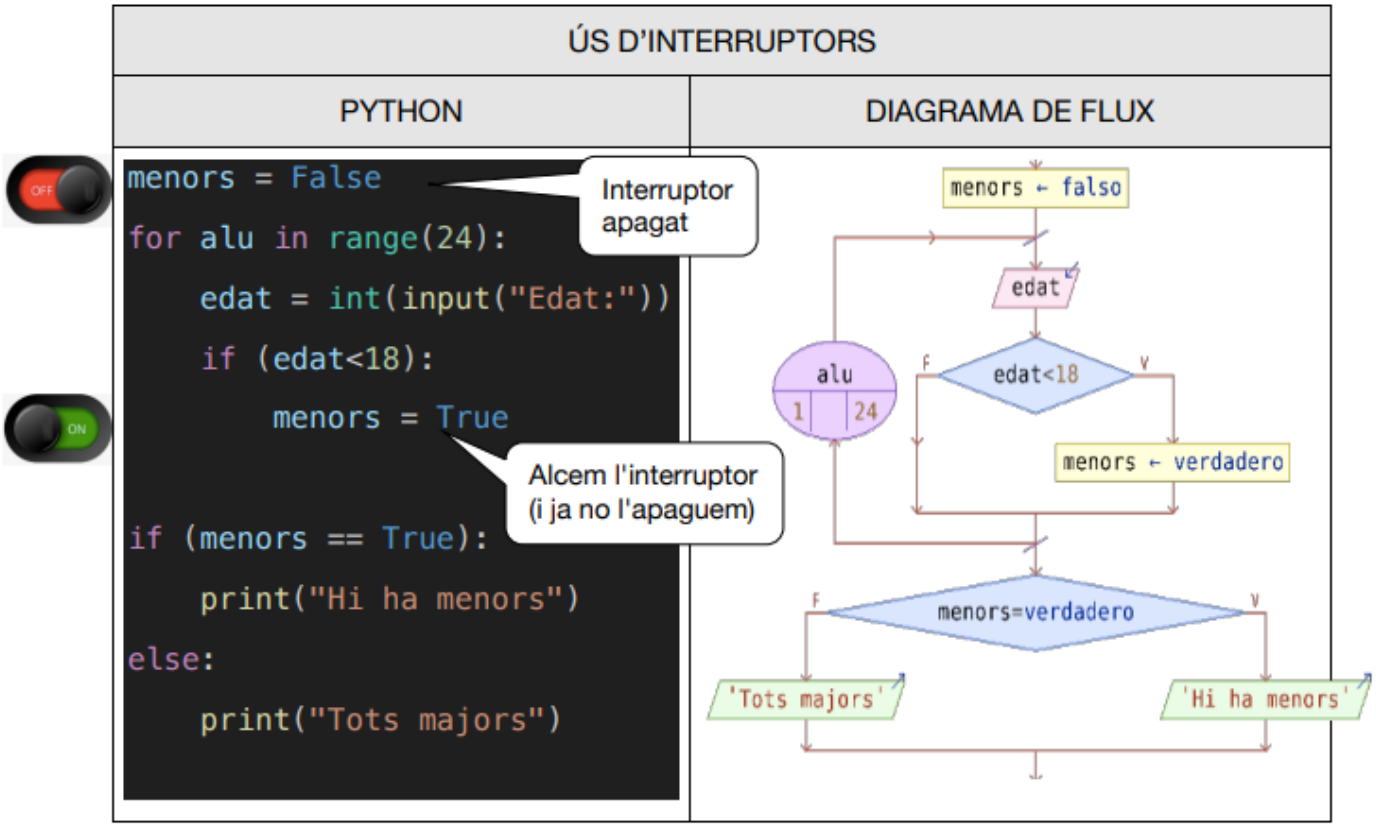
Exercici 34

Calcula el factorial d'un número introduït per teclat.

Interruptors

Els *interruptors* (també coneguts com *indicadors*, *banderes* o *flags*) són variables destinades a indicar si en alguna de les iteracions d'un bucle **ha passat o no** una cosa determinada.

Estes variables seran de tipus **lògic** (booleà) ja que només guardaran 2 possibles valors: ha passat alguna cosa (vertader) o no (fals). En *Python*, estos valors són *True* i *False* (en *PSeInt* són *Verdadero* i *Falso*). Ja vorem que en *Java* són *true* i *false* (en minúscula).



Però si un llenguatge no admetera el tipus booleà (com el llenguatge C), podríem usar una variable de tipus enter (per exemple usant els valors 0 i 1) o de tipus caràcter (per exemple emprant els valors "V" i "F", o bé, "S" i "N", etc).

Exercici 35. Ús d'interruptors

Demana un número per teclat i digues si és un número primer o no. Un número és primer si només té 2 divisors (per ell mateix i per 1, clar).

2.5 Alguns algorismes bàsics

Obtindre el major d'una llista de números

Imaginem que (sense cap ordinador), vull anar preguntant l'edat de tot l'alumnat per a poder saber l'edat màxima. Com ho faria? No he de recordar l'edat de tots, sinó que només necessite saber en cada moment l'edat de l'alumne actual i l'edat màxima obtinguda fins eixe moment (que hauré d'anar canviant o no).

Per tant, necessite 2 variables: *edat* (per a guardar l'edat de l'alumne "actual") i *maxima* (per a guardar l'edat màxima fins a eixe moment). L'algorisme seria:

Algorisme

• Inicialitzar *maxima* a un valor molt xicotet. En el cas de les edats ens serviria el 0, ja que cap alumne pot tindre menys de 0 anys.

• Per cada alumne del bucle faria:

• Demanar l'edat de l'alumne (la guarde sempre a la variable *edat*).

• Si eixa edat és major que la que tinc en *maxima*, canvie el valor de *maxima* a eixa edat.

```
maxima = 0

for i in range (24):
    edat = int(input("Edat: "))
    if (edat > maxima):
        maxima = edat

print ("Edat màxima:", maxima)
```

Ara bé: i si no foren edats? És a dir: imaginem que volem calcular el número màxim d'una llista de números però que també podrien ser negatius. Fins i tot, tots els números podrien ser negatius. En eixe cas no ens serviria inicialitzar el valor de la variable *maxima* a 0, sinó que hauríem d'assignar-li un valor que siga més xicotet que tots els que puguen estar en la llista.

En eixe cas tenim 2 opcions per a trobar el valor inicial de la variable *maxima*:

1. El valor més xicotet que admet eixe tipus de dades en eixe llenguatge. Per exemple, el menor enter en *Java* seria *Integer.MIN_VALUE*. Compte! *Python* no té límit en la grandària dels enters.
2. El primer valor de la llista de números.

Per tant, si volem obtindre el màxima de 100 valors usant l'opció (2), al principi de l'algorisme direm que el màxim és el primer d'eixos valors. Després llegirem en un bucle els altres 99 números i, com abans, anirem canviant, si cal, el valor màxim:

```
max = int(input("Núm: "))

for i in range (99):
    num = int(input("Núm: "))
    if (num > max):
        max = num

print("Màxim:", max)
```

Exercici 36. Càlcul del major

Llig uns quants números (fins que posem 0). Mostra el major i el menor.

Bucles niuats

Podem posar un bucle (o més) dins d'un altre.

Per exemple, si volem obtindre de teclat un número imparell, cal un bucle per a demanar contínuament el número fins que siga imparell. Però si volem demanar **4** números imparells, haurem de posar el bucle anterior dins d'un altre bucle:

Bucles niuats

```
for in range (1, 5):
    num = int(input("Núm. imparell:"))

    while num % 2 == 0:
        print("No és imparell. Torna")
        num = int(input("num imparell: "))

    print(f"El {num} és imparell")
    print(f"Ja tenim {i} imparells")
```

NOTA: recorda que l'operador `%` calcula el *residu de la divisió entera*, tant en *PSeInt*, *Python* i *Java* (encara que *PSeInt* també admet l'operador *mod*).

Exercici 36.quadrat

Dibuixa un quadrat de caràcters `x`, de grandària `n` (demanada per teclat). Per exemple, si `n` és 4, cal dibuixar:

```
x  x  x  x
x  x  x  x
x  x  x  x
x  x  x  x
```

NOTA: per a escriure sense que faça després un salt de línia:

```
print("Hola", end="")
```

Exercici 36.triangle

Dibuixa un triangle de caràcters `x` d'altura `n` (demanada per teclat). Per exemple, si `n` és 4:

```
x
x  x
x  x  x
x  x  x  x
```

Exercici 36.triangle2

Dibuixa un triangle de `x` d'altura `n` (demanada per teclat). Per exemple, si `n` és 4:

```
x  x  x  x
x  x  x
x  x
x
```

Exercici 36.rectangle

Dibuixa un rectangle de `x` de grandària `alt` per `ample` (dades demanades per teclat). Per exemple, si és 4 d'alt per 7 d'ample, dibuixarem:

```
x  x  x  x  x  x  x
x  x  x  x  x  x  x
x  x  x  x  x  x  x
x  x  x  x  x  x  x
```

Exercici 36.rectangleBuit

Igual que el 36.4, però ara el triangle ha d'estar buit:

```
x  x  x  x  x  x  x
x                 x
x                 x
x  x  x  x  x  x  x
```

Exercici 37

Mostra les taules de multiplicar del 2 al 9.

PISTA: ja havies fet, amb un bucle, una taula de multiplicar. Ara es tracta de posar eixe trós de codi dins d'un altre bucle, ja que volem *motes* taules.

Exercici 37.1

Demana un número per teclat (n) i mostra per pantalla les següents línies:

```
1 = 1
1 + 2 = 3
1 + 2 + 3 = 6
...
fins a les n línies
```

Exercici 37.2

Mostra els primers 15 números primers. Els resultat ha de ser:

```
Nombres primers:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47
```

NOTA: L'1 no és primer, per definició.

2.6 Exercicis

Exercici 38

En un pàrquing es paga 2€ l'hora completa. I per als minuts restants, es paga 4 cèntims per minut, però no pot excedir el preu d'una hora. Calcula què li toca pagar a un conductor per un determinat temps, donat en minuts.

NOTA: es poden eliminar els decimals amb la funció `trunc`.

Exercici 39

Donat un temps en segons, calcula els segons que falten per a convertir-se en minuts sencers. Per exemple, per a un temps de 70 segons, en faltarien 50.

Exercici 40

Donat un temps en minuts, calcula els dies, hores i minuts que li corresponen.

Exercici 41

Pregunta quina és l'arrel quadrada de 225 fins que siga encertat.

Exercici 42

Càlcul del sou d'un treballador. Pregunta quantes hores ha treballat (a partir de les 40 hores es consideraran extres). Pregunta quin és el preu de l'hora normal (el preu de l'hora extra és un 50% més del preu de l'hora normal). Que cobra el treballador?

Exercici 43

Modifica l'exercici de les hores extres per a obtindre la suma dels salaris de tots els treballadors. Tots la mateixa tarifa. Acabarà quan posem 0 hores treballades.

Exercici 44

Demana `N` notes d'un estudiant i calcula:

• Quantes notes té suspeses.

• Quantes aprovades.

• La mitjana de les notes.

• La mitjana de les notes aprovades i la mitjana de les suspeses.

Exercici 45

Escriu un algorisme que calcule el total d'una factura d'un article determinat del qual s'adquirixen `N` unitats a un preu `P`. L'IVA és el 21%. Si l'import a pagar (amb IVA) és superior a 300€, s'aplicarà un descompte del 5%.

Exercici 46

Modifica l'exercici de la factura per a demanar moltes voltes el preu i la quantitat de diferents articles. Si les unitats introduïdes són 0, vol dir que no es demanaran més articles. El possible descompte, s'aplicarà al final de la factura.

Exercici 47

Algorisme que vaja demanant lletres des de teclat i que pare quan es trobe amb una vocal. Cal escriure esta vocal.

Exercici 48

Mostra els números entre 100 i 200 que són múltiples de 3 però que no son múltiples de 2.

Exercici 49

Mostra tots els divisors d'un número donat, Utilitzant l'operador `mod`.

Exercici 50

Demana per teclat la quantitat del números que vol introduir l'usuari. A continuació, llig de teclat eixa quantitat de números i digues de cadascun si és parell i positiu al mateix temps.

Exercici 51

Imprimix la taula de multiplicar d'un número introduït per teclat.

Exercici 52

Dir si un número de 3 xifres és o no d'Armstrong (si és igual a la suma dels seus dígits al cub, per exemple: $153 = 1^3 + 5^3 + 3^3$). Si no ho és, torna-ho a intentar per a més números.

Exercici 53

Algorisme que mostre tots els números d'Armstrong de l'1 al 1000.

Exercici 54

Comprova si un número donat és perfecte.

NOTA: es diu que un número és perfecte si és igual a la suma dels seus divisors excepte ell mateix.

Exercici 55

Mostra els números primers menors de 100.

2.7 Activitat obligatòria

Partit de bàsquet

A l'institut es realitzarà un partit de bàsquet, i volem anar anotant en temps real els punts de cada cistella que es fa.

Així podrem saber com van en cada moment i podrem traure una xicoteta estadística després de cada quart i en acabar el partit.

Activitat

Fes un programa que faça el que es detalla a continuació, tenint en compte que has d'usar, on calga, sentències de bifurcació doble o múltiple, bucles de tipus condicional i incondicional, i comptadors i acumuladors.

L'algorisme és el següent

• Demana per teclat els noms dels 2 equips

• Per cadascun dels 4 quarts del partit caldrà fer:

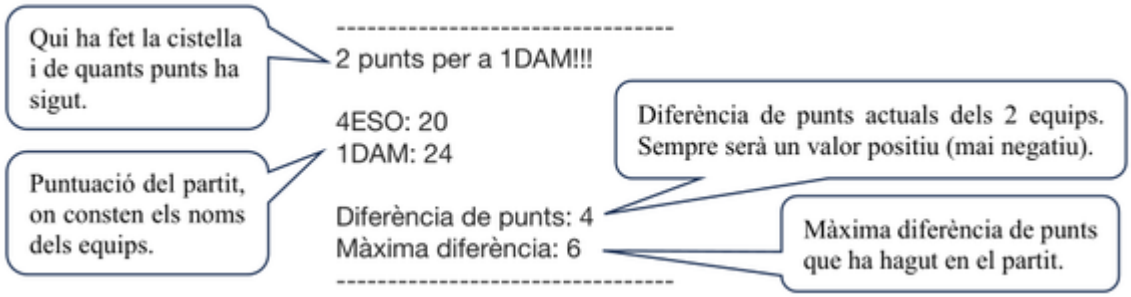
• Mostrar informació del quart que comença. Per exemple: `-- COMENÇA EL QUART NÚMERO 1 --`

• Bucle que registre cadascuna de les cistelles anotades en eixe quart. S'aniran demanant punts de les cistelles efectuades fins que s'introduïsca la puntuació de 0 (que servirà per acabar el quart). És a dir: per cada cistella efectuada caldrà fer el següent:

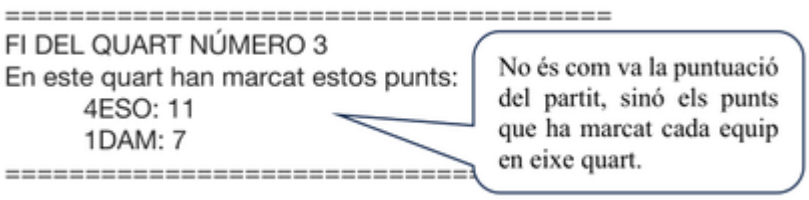
• Preguntar de quants punts ha sigut la cistella (o bé, un 0 per a acabar el quart). Caldrà controlar que el valor introduït estiga entre 0 i 3.

• Preguntar quin equip ha fet la cistella (A o B). Caldrà controlar que el valor introduït siga una A o una B.

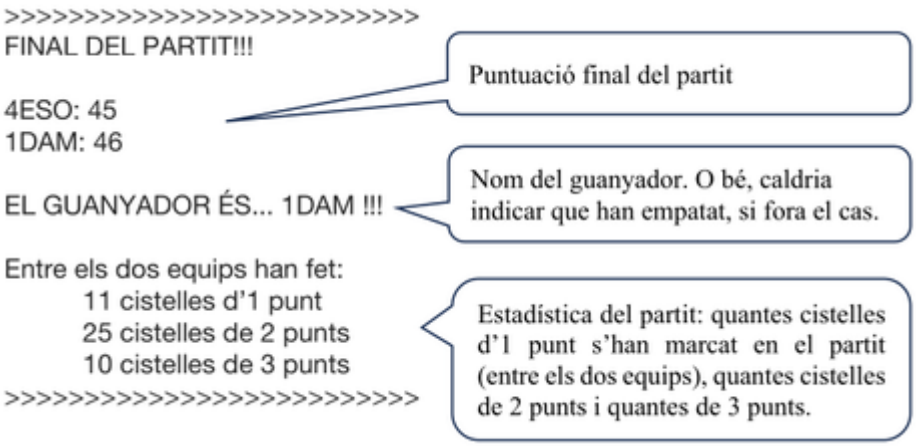
• Mostrar com va la puntuació del partit.



• En acabar cada quart es mostrarà l'estadística **d'eixe quart**.



• En acabar el partit es mostrarà el resum del partit.



NOTES:

- Per si no ha quedat clar què ha de fer el programa, ací teniui al nostre company Abdó, explicant-vos l'execució d'aquest.
- El treball és individual. Podeu consultar-me a mi tot el que vullgau, però no als companys ni a *ChatGPT* ni similar. Si detectem que algú s'ha copiat alguna part, tindran un 0 els dos (copiat i copiador). El treball està pensat més que res perquè aneu ben preparats a l'examen. De fet, a l'examen podria preguntar-vos alguna modificació del vostre programa, etc.
- Cal fer l'algorisme en *Python*. El nom del fitxer es dirà:
- ***basquetGarciaPep.py*** (si et digueren Pep Garcia, clar)
- Posa el teu nom complet com a comentari al principi del programa
- Posa algún comentari més pel programa (sense passar-te'n).
- Puja el fitxer a Aules.

3. UD3. Introducció a Python

Al tema anterior, per començar a vore els principis de la programació estructurada hem vist algorismes en ordinograma i la seua implementació en Python. En este tema aprofundirem un poc més en Python pel que fa als operadors i expressions, així com a l'entrada i eixida de dades. A més, com que més endavant es vorà el llenguatge Java, conforme anem veient coses en Python, anirem dient ja les diferències en java. Estes diferències solen ser iguals en Java i en C.

3.1 Característiques bàsiques de Python

Python és un llenguatge creat a finals dels 80 per *Guido Van Rossum*, i deu el seu nom a l'afició pel grup còmic britànic *Monty Python*.

És un llenguatge de programació **interpretat**, de **tipificació dinàmica** i **multi plataforma**. És de propòsit general, fins i tot per a la creació de *scripts*.

Què necessitem per a programar en Python?

Ho podem fer de diverses maneres:

- Usar un intèrpret online (no cal instal·lar res)

  Per exemple, onlinegdb.com. Caldrà seleccionar el llenguatge *Python 3*, escriure el codi i polsar *Run*.
- Instal·lar l'intèrpret de Python: www.python.org/downloads

  Una vegada instal·lat (Linux ja el porta) podrem:
- Provar comandaments en mode interactiu: en el terminal escriurem ***python3***. Entrarem en el mode interactiu: després de fer *intro* en cada instrucció, s'executarà.

```
paco@tufdash:~$ python3
Python 3.12.3 (main, Sep 11 2024, 14:17:37) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 10
>>> b = 20
>>> print(a+b)
30
>>> print(a*b)
200
>>> quit()
paco@tufdash:~$
```

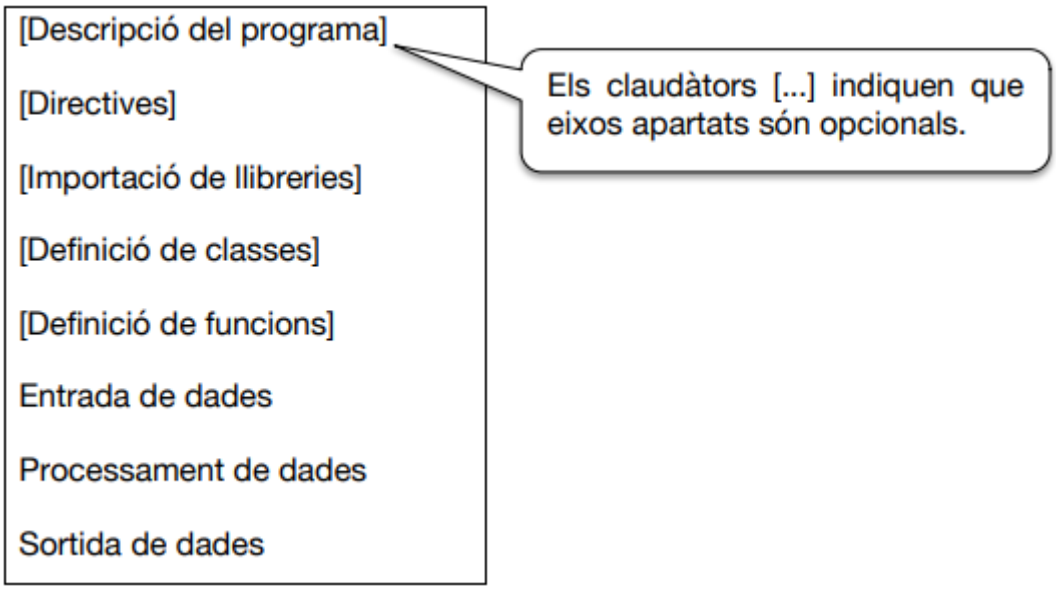
- Escriure el programa en un fitxer de text: Li direm, per exemple, *holaMon.py*, i l'executarem amb: ***python3 holaMon.py***. No cal compilar res, perquè Python és interpretat.
- Usar un IDE (Entorn de Desenvolupament Integrat): Per exemple, Visual Studio Code. Potser calga instal·lar l'extensió de Python. Escriurem el nostre programa, el guardarem (***holaMon.py***) i l'executarem amb la icona del triangle.

Estructura d'un programa en Python

Un programa en Python té una estructura molt simple:

- No s'ha de posar punt i coma després de cada instrucció.
- No es posen delimitadors de bloc de programa. Simplement se sangra.
- No cal indicar el tipus de dades de les variables.

Qualsevol programa escrit en Pyhton té la següent estructura:



Veiem que l'única part (pràcticament) necessària en un programa és l'entrada, processament i la sortida de dades, com en qualsevol llenguatge de programació.

En Python estos apartats poden variar de posició.

Veiem uns exemples i els analitzem.

L'exemple més simple, mostrar un missatge per pantalla:

```
print("Hola, Pep")
```

Un altre exemple més complet:

```
# Programa que calcula l'àrea d'un rectangle (1)
# Autor: Pep Garcia
# Data: 2024/10/01

import time  #(2)!

def areaRectangle(base, altura): # (3)!
    return base * altura

b = int(input("Dis-me la base del rectangle: ")) #(4)!
a = int(input("Dis-me l'altura del rectangle: "))

time.sleep(2) # Espera 2 segons

area = areaRectangle(b, a) #(5)!

print("L'àrea del rectangle és", str(area)) #(6)!
```

1. Descripció del programa
2. Importació de llibreries
3. Definició de funcions
4. Entrada de dades
5. Processament de dades
6. Sortida de dades

Exemple

Copia i apega el codi anterior en l'IDE que tingues de Python i executa'l per vore què fa.

Comentem, per adamunt, algunes coses del programa que més endavant ampliarem:

- Descipció del programa. Son comentaris, no s'executen.
- A l'*import* indiquem que necessitem una llibreria: conjunt de funcions que ja estan implementades i les podem fer servir als nostres programes. En este exemple la llibreria és *time* i ens cal per a usar la funció *time.sleep()* que fa que el prgrama pare en eixe punt uns segons (2 en este cas).
- Després tenim la definició d'una funció, que comença amb la paraula reservada *def*. Tot el que es pose dins del bloc de la definició d'una funció ha d'anar sagnat (en este cas és només la instrucció *return*).
- Després ja tenim el nostre programa pròpiament dit, on veiem que:
- Les línies no tenen cap sagnat (van just a l'esquerra)
- Amb els *input* aconseguim que s'introduisquen dades per teclat.
- Es fa la crida a la funció que hem definit abans (*areaRectangle*).
- Finalment mostrem a l'usuari el resultat (*print*).

Noms de variable i funcions en Python

Abans hem vist que hem posat noms de variable i funcions. Estos noms han de tindre unes **regles**, que solen ser les mateixes en tots els llenguatges, encara que poden variar un poc. En Python estes són les regles per als noms de variables i funcions:

Regles per a noms de variables i funcions en Python

- Son una combinació de lletres minúscules [a...z], majúscules [A...Z], dígiint [0...9] i el caràcter subratllat [_].
- Poden tindre qualsevol longitud.
- S'admeten els accents, la ç i la ñ.
- No poden haver símbols especials ni operadors: `[, !, @, #, $, %, *, ...`
- No poden començar amb dígit.
- No poden ser paraules reservades:

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Variables

Les variables són els llocs on es guarda la informació (per exemple, els llocs on es guarda cada dada introduïda per teclat).

Es poden classificar en globals i locals:

- **Variables globals**: es creen fora de qualsevol funció. Es pot accedir a elles des de qualsevol part del programa.
- **Variables locals**: es creen dins d'una funció. Es pot accedir a elles només des d'eixa funció.

Més endavant vorem els tipus de dades (enter, caràcter, etc) de les variables.

Comentaris

En algunes parts del programa cal que el programador pose anotacions per a:

- Recordar el que ha fet, per a futures modificacions.
- Indicar a altres programadors com s'ha fet alguna cosa.
- Indicar la data (o autor, etc.) de creació del programa.

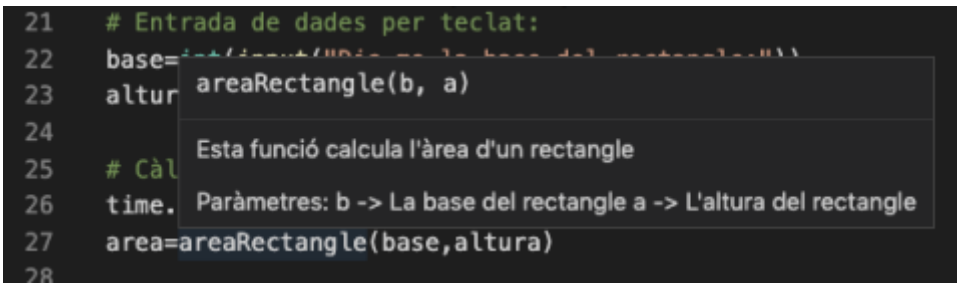
Tipus de comentaris:

- **D'una línia**:
- Precedit per coixinet: # soc un comentari
- Entre cometes simples: 'soc un comentari'
- Entre cometes dobles: "soc un comentari"
- **De diverses línies**:
- entre trios de cometes simples: ' ' '
- entre trios de cometes dobles: " " "
- **De documentació de funcions**: si posem un comentari entre cometes en la primera línia dins d'una funció (per a explicar què fa) després podrem accedir a eixe comentari des d'altres parts del programa. Ara ho veurem.

Veiem en este programa exemples dels diferents tipus de comentaris:

```
'''
Programa que calcula l'area d'un rectangle
Autor: Pep Garcia
Data: 1-10-2024
'''
import time
# ----------------------------------------
def areaRectangle(b, a):
    '''
    Esta funció calcula l'àrea d'un rectangle
    Paràmetres:
        b -> La base del rectangle
        a -> L'altura del rectangle
    '''
    return b*a
# ----------------------------------------

# Entrada de dades per teclat:
base = int(input("Dis-me la base del rectangle: "))
altura = int(input("Dis-me l'altura del rectangle: "))

# Càlculs
time.sleep(2) # Espera dos segons
area = areaRectangle(base, altura)

# Eixida de resultats per pantalla:
print("L'àrea del rectangle és", str(area))
```

Ens alguns IDE com el *Visual Studio Code*, si poses el cursor damunt del nom d'una funció et mostra els comentaris que has posat en ella:



Delimitacions

Son símbols especials que permeten al compilador reconéixer les diferents parts del programa.

El més important és el finalitzador de sentències, que, en molts llenguatges de programació (com C i Java) és el pun i coma (;) però Python fa servir simplement el bot de línia.

Ací tenim els delimitadors que s'usen en Python.

Python	C i Java	Nom	Utilitat
Salt de línia	;	Finalitzador	- Finalitzar una instrucció simple o una declaració de variables
Tabulació	{ }	Bloc	- Delimitar inici i fi d'un bloc de codi
,	,	Separador	- Separar els elements d'una llista
( )	( )	Parèntesi	- Agrupar operacions - Paràmetres de funcioins
[ ]	[ ]	Claudàtors	- Per a vectors, llistes...

3.2 Tipus de dades

Les dades que manegen els programes són de distints **tipus**: lletres, números sense decimals, amb decimals...

Per tant, les variables seran d'un tipus determinat. En la majoria de llenguatges de programació, abans d'utilitzar una nova variable, cal definir-la (declarar-la): indicar de quin tipus és. Però en Python no cal. Simplement el tipus de la variable serà del mateix tipus que el valor que li s'assigne.

```
edat = 30
nom = "Pep"
pes = 74.5
casat = True
```

Veiem els distints tipus que solen tindre els llenguatges de programació.

Tipus elementals

En Python hi ha 4 tipus bàsics: enter, amb decimals, cadena i lògic.

Altres llenguatges, com C i Java, en tenen més, per a indicar enters xicotets o grans, amb signa o sense... Igual que per a números amb decimals.

Números enters: *int*

Números enters (sense decimals).

Quan posem un número s'interpreta que està en sistema de numeració decimal. Però podem dir-li que ho interprete com a binari, octal o hexadecimal:

```
print(11)   # Número 11 en sistema decimal. Mostra 11
print(0b11) # Numero 11 en sistema binari. Mostra 3
print(0o11) # Número 11 en sistema octal. ostra 9
print(0x11) # Número 11 en sistema hexadecimal. Mostra 17
```

El *print* mostra el número en sistema decimal.

Números amb decimals: *float*

Números amb decimals

```
print(5.2)  # mostra 5.2
print(5.)   # mostra 5.0
print(.2)   # mostra 0.2
print(5e2)  # mostra 500
print(5e-2) # mostra 0.05
```

Lògics: *bool*

Serveix per si una variable volem que tinga 2 únics estats (vertader o fols). Els únics valors que pot tindre una variable d'este tipus són *True* o *False*.

```
majorEdat = True
jubilat = False
```

Ens servirà per a quan usem sentències condicionals (ja entrarem en detall):

```
if jubilat:
    print("Està jubilat")
else:
    print("No està jubilat")
```

NOTA

En C++ i Java és *true* i *false* (en minúscula). C no té eixe tipus de dades com a tal (usa el 0 per a *false* i l'1 per a *true*).

Cadenes: *str*

És el tipus de dades per a guardar una cadena de caràcters (un nom de persona, per exemple). Una dada de tipus *str* és una successió de 0 o més caràcters dins de cometes simples o dobles (encara que es recomana entre cometes dobles, ja que molts llenguatges només admeten les dobles).

```
nom = "Pep Garcia"
domicili = 'Carrer La Punt, 54'
print("Nom de l'alumne:", nom)
```

Per a guardar cadenes els llenguatges utilitzen formes distintes:

- En **C** no és un tipus sinó un **vector** de caràcters (ja vorem els vectors).
- En **Java** no és un tipus, sinó una **classe** (ja vorem les classes).
- En **Python** sí que és un **tipus** de dades.

SEQÜÈNCIES D'ESCAPAMENT

Per a poder posar unes cometes dobles dis d'una cadena amb cometes dobles es pot usar el caràcter d'escapament `\`. També per a cometes simples:

```
print("Podem mostrar 'i \" dins de cometes dobles")
print('Podem mostrar \' i " dins de cometes simples')
```

En una cadena de text també podem utilitzar este caràcter d'escapament per a representar diverses accions:

Python, C i Java	Acció	Exemple	Resultat
\n	Nova línia	print("Hola\nAdéu")	
\t	Tabulador	print("Hola\tAdéu")	Hola Adèu
\r	Retorn de carro	print("Hola\rTu")	Tula
\b	Backspace	print("Hola\bAdéu")	HolAdéu

Encara que les més emprades són `\n` i `\t`.

Totes estes seqüències d'escapament també es poden usar en C i Java.

Tipus composts

Els tipus simples (que acabem de veure) serveixen quan hem de guardar una informació simple. És a dir, formada per una sola dada (una temperaura, un nom, una edat...)

Però si volem guardar en una variable el domicili d'un client (format per un carrer, número, codi postal...) o una data (dia, mes any), el programador haurà de definir un tipus de dades compost. Estos tipus de dades compostos els vorem més endavant.

Declaració de variables

Una variable és una porció de memòria (RAM), representada per un nom (identificador) on es guardarà un valor que pot variar al llarg de l'execució d'un programa.

Declarar una variable vol dir indicar de quin tipus serà eixa variable. Depenent del llenguatge de programació caldrà declarar les variables o no. Per tant, atenent a este criteri, tenim 2 tipus de llenguatges de programació:

Llenguatges de tipificació estàtica (C, Java... però no Python)

Estos llenguatges **obliguen a indicar de quin tipus serà una variable** abans d'usar-la. Després, al moment de fer servir les variables, el llenguatge controla que el valor que s'assigne a una variable corresponga al tipus de la variable. Si no és el cas, donarà error-

```
int n;
float x;
n = 3;
n = "Pep";  // Error perquè no és del mateix tipus.
n = 4.7;    // No dóna error però lleva la part decimal. Guardarà un 4.
z = 6;      // Error perquè la variable z no està declarada prèviament.
```

Llenguatges de tipificació dinàmica (Python, PHP...)

En estos llenguatges **no es declara la variable** prèviament. Simplement quan se li assigna un valor, la variable agafa el tipus d'eixe valor. I pot variar de tipus cada vegada que se li assigna un nou valor.

```
n = 7       # Primera vegada que ix la variable n. n val 7 i és int
n = 5.67    # Ara n val 5.67. Per tant, ara és float
n = 9       # Ara n val 9. Per tant, torna a ser int
n = n+2     # Ara n val 11. Continua sent int
n = n/4     # Ara n val 2.75. Per tant, ara n és float
n = "Pep"   # Ara n és str (cadena)
n = n+2     # ERROR. No li podem sumar 2 al text "Pep"
```

Com veiem a l'exemple, no tindrem les situacions d'error dels llenguatges de tipificació estàtica (ja que no s'ha de declarar la variable i poden canviar de tipus). Però pot ser un desavantatge ja que podria ser que volguérem que en fer la divisió de `11/4` volguérem guardar la part entera (2) i no 2.75. Ens pot portar a situacions inesperades o inconsistents. Caldrà anar en compte en estos casos.

> **NOTA**
>
> Si en algun moment parlem de "declarar" una variable en Python, ens estarem referint al primer moment del programa on li s'assigna un valor a eixa variable.

Àmbit i visibilitat

> **NOTA**
>
> Aquestos conceptes s'explicaran en detall quan veiem la programació modular (funcions). No obstant, veiem un avanç.

Les variables poden "declarar-se" (començar a usar-se) en qualsevol part del programa, però segons el lloc on siguen declarades, les podrem fer servir només en alguna part (**variables locals**) o bé en tot el programa (**variables globals**).

Exemple 1

La variable `a` és **local** a la funció "funcioneta"

```
# ---------------------
def funcioneta():
    a = 3 # declarem la variable 'a' com a LOCAL (dins de la "funcioneta")
    print(a)
# ---------------------

print("El programa comença a executar-se per ací")
funcioneta()
print(a) # ERROR perquè accedim a la variable local de "funcioneta"
```

Si una variable està declarada dins d'una funció, només podem accedir a ella dins d'eixa funció.

Exemple 2

```
# ------------------------
def funcioneta():
    print(a)
# ------------------------

print("El programa comença a executar-se per ací")
a = 3   # Declarem la variable 'a' com a GLOBAL (fora de les funcions)
funcioneta()
print(a)
```

Si una variable "delarada" (primer ús) fora de les funcions, podrem accedir a ella des de qualsevol lloc del programa (bé, sempre després de ser declarada). Però ja vorem que no convé declarar variables globals.

L'àmbit i visibilitat d'una variable són conceptes íntimament relacionats. Fan referència a des d'on es pot accedir a una variable:

- La **visibilitat** és la propietat que indica si es pot accedir o no a una variable en un punt determinat del programa.
- En l'exemple 1:
- Dins de la funció "funcioneta" sí que hi ha visibilitat de 'a'.
- Fora de la funció "funcioneta" no hi ha visibilitat de 'a'.
- En l'exemple 2:
- En tot el programa sí que hi ha visibilitat de 'a'.
- L'**àmbit** és la zona del programa on és visible una variable.
- En l'exemple 1, l'àmbit de 'a' és dins de la funció "funcioneta"
- En l'exemple 2, l'àmbit de 'a' és a tot el programa.

En **C** i **Java**, a més de definir variables locals a una funció es poden definir locals a un bloc de codi, tancat entre claus `{}`. En eixe cas, eixes variables només poden ser accedides dins d'eixe bloc de codi.

En **Python** un bloc seria el trós de codi (seguit) amb el mateix sagnat (o subsagnat). Però si una variable es declara en eixe bloc, en Python sí que podem accedir des d'altres blocs, encara que no és recomanable:

```
if (edat >= 18):
    major = True #(1)!
else:
    major = False

print(major) #(2)!
```

1. El primer ús de "major" es fa en este bloc
2. Però Python em permet usar-la fora del bloc

En canvi, és recomanable declarar la variable en el "bloc de fora":

```
major = False #(1)!

if (edat >= 18):
    major = True

print(major) #(2)!
```

1. Declarem la variable en el "bloc de fora"
2. ... ja que vaig a usar-la en eixe bloc

3.3 Operadors

Anem a veure els distints operadors que solen tindre els llenguatges de programació i a contruir expressions amb elles, així com la forma d'introduir dades per teclat i mostrar resultats per pantalla.

Operadors aritmètics

Python	C i Java	Significat	Observacions
+	+	Suma	
-	-	Resta o signe	
*	*	Multiplicació	
/	/	Divisió amb decimals (11/4 -> 2.75)	En C i Java: (11/4 -> 2 -- 11/4.0 -> 2.75)
//	NO	Divisió entera (11//4 -> 2)	
%	%	Residu divisió entera (21%4 -> 1)	
**	NO	Potència (2**3 -> 8)	

Operadors relacionals

Python	C i Java	Significat
==	==	Igual
!= <>	!=	Distint
<	<	Menor
<=	<=	Menor o igual
>	>	Major
>=	>=	Major o igual

Ja veurem que, principalment, estos operadors s'utilitzen en les condicions de les instruccions *if* i *while*.

Exemples

```
if (nota >= 5):
    print("Aprovat")
...
```

```
while (edat < 0):
    print("Edat incorrecta. Torna-me-la a dir")
...
```

Operadors lògics

Ja veiérem al primer tema quins eren els operador l'ogics i com actuaven (recordeu les "taules de veritat"). Veiem ara com es representen en Python, C i Java:

Python	C i Java	Significat
or	||	Vertader si algun és vertader
and	&&	Vertader si els 2 són vertaders
not	!	El contrari

Exemples d'operadors lògics

```
...
if edat >= 18 and edat <= 65:
    print("Estàs en edat de treballar")
else:
    print("No estàs en edat de treballar")
...
```

```
...
while nota < 0 or nota > 10:
    print("Nota incorrecta. Torna-la a posar")
...
```

```
plou = True
faSol = False
print(plou or faSol)    # True
print(plou and faSol)   # False
print(not plou)         # False
```

Curtcircuit d'expressions

Si recordem les taules de veritat, podem afirmar que...

- *false* AND ... --> ***false***
- *true* OR ... --> ***true***

Per tant, com les expressions s'avaluen d'esquerra a dreta, en el moment en què el compilador puga assegurar el valor final de l'expressió lògica (*True* o *False*), pararà d'avaluar-la. Esta manera de treballar s'anomena *curtcircuit d'expressions*. Això ens dóna un benefici pel que fa al control d'errors i a la valocitat d'execució.

Exemples

```
if (descompte1 > 0 or descompte2 > 0 or descompte3 > 0):
    print("S'aplica algun descompte")
```

Si el `descompte1` és major que `0`, ja no es comproven les altres 2 expressions i passa a executar-se directament el *print*.

```
(x<0) and print("El valor de la variable és negatiu")
```

Només mostrarà el text si el valor de x és negatiu.

Exercici sobre el curtcircuit d'expressions

Què passarà en cada cas?

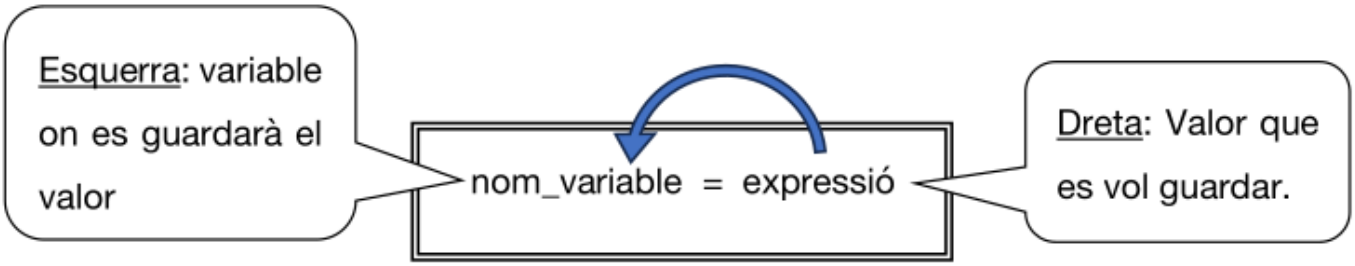
```
sumaNotes = 50
quantsAlumnes = 0
if (quantsAlumnes > 0) and (sumaNotes/quantAlumnes >= 5):
    print("Nota miyjana aprovada")
else:
    print("Nota mitjana no aprovada")
```

```
if (sumaNotes/quantAlumnes >= 5) and (auntAlumnes > 0)
```

Operador d'assignació

Este operador ja ha aparegut en molts exemples. S'utilitza quan volem emmagatzemar un valor en una variable. En Python i en la majoria de llenguatges de programació, l'operador d'assignació és el símbol *igual (=)*.

Python	C i Java	Significat
=	=	Assignació



És a dir: primer s'avalua la part de la dreta, i després s'assigna eixe resultat a la variable esquerra.

Exemple

```
x = 10      # x valdrà 10
y = 20      # y valdrà 20
x = x + 1   # x no valdrà 10, sinó 11
y = x + y   # y no valdrà 20, sinó 31
```

Per descomptat, l'assignació és *destructiva* (com es veu a l'exemple): sempre que es fa una assignació elimina el valor antic de la variable. És a dir, només pot guardar una dada en un moment determinat.

Si volem assignar un mateix valor a moltes variables també ho podem fer així (també en C i Java):

```
a = b = c = d = 10
```

Operadors aritmètics reduïts (operadors aritmètics i d'assignació)

L'operador d'assignació que hem vist (=) assigna un valor a una variable. Però si el que volem fer és augmentar (o disminuir) el valor que ja té la variable, podem usar els operadors aritmètics reduïts.

Python	C i Java	Significat	Assignació reduïda	Assignació equivalent
+=	+=	Suma i assignació	x += y	x = x + y
-=	-=	Resta i assignació	x -= y	x = x - y
*=	*=	Producte i assignació	x *= y	x = x * y
**=	NO	Potència i assignació	x **= y	x = x ** y
/=	/=	Divisió i assignació	x /= y	x = x / y
%=	%=	Residu i assignació	x %= y	x = x % y
//=	NO	Divisió entera i assignació	x //= y	x = x // y
NO	++	Autoincrement	x++	x = x + 1
NO	--	Autodecrement	x--	x = x - 1

Com veiem, estos dos operadors fan 2 coses: una operació aritmètica i una assignació. També es coneixen com *operadors d'actualització*. En les dos columnes de la dreta, `x` és una variable , i `y` és una expressió, constant o variable.

Exemple

```
y = 1
x = 4
x += y      # x = x + y         ---> x = 5
x *= 2      # x = x * 2         ---> x = 10
x -= 3 - y  # x = x - (3 - y)   ---> x = 10 - (2)   ---> x = 8
```

Exercici 2. Assignacions

Al següent programa Python, què valdrà cada variable després de cada assignació?

```
a = 6
b = 3
b = 1 + b          # a =      b =
a = a / b          # a =      b =
b = 6 // b + b     # a =      b =
```

Exercici 3

En el següent programa Python, què valdrà cada variable després de cada assignació?

```
a=4
b = 20
b += 23     # a=     b=
b //= 2     # a=     b=
a -= 2      # a=     b=
a *= b + 2  # a=     b=
a %= b      # a=     b=
a**= b-19   # a=     b=
```

Altres operadors

El *sizeof*

La quantitat de *bytes* que s'utilitza per a guardar una dada depén del tipus de dades, així com del llenguatge de programació, la varsió del compilador i del tipus de processador que s'utilitze (32 o 64 bits).

Per tant, perquè el nostre programa puga ser portable, de vegades és necessari saber quants *bytes* ocupen les variables amb les quals treballarem. Per això alguns llenguatges tenen una funció per a tal fi:

```
import sys
x = 10
text = "10"
print(sys.getsizeof(x), "bytes")            # 28 bytes (1)
print(sys.getsizeof(text), "bytes")         # 51 bytes (2)
print(sys.getsizeof(1000), "bytes")         # 28 bytes
print(sys.getsizeof("Hola, Pep"), "bytes")  # 58 bytes
print(sys.getsizeof(1234.56789), "bytes")   # 24 bytes (3)
```

1. Qualsevol enter ocupa 28 bytes
2. Quantitat de caracters + 49
3. Un float ocupa 24 bytes

3.4 Expressions

Ja hem vist les dades i els operadors. Amb ells podem formar expressions.

Una expressió és una combinació de dades (operands) i operadors (seguint certes regles de construcció) i que poden estar agrupats per parèntesi per a indicar l'ordre de càlcul.

Exemples d'expressions

• base * altura / 2
• (-b + sqrt(b*2* - *4*ac))/2
• 1000
• edat >= 0 and edat <= 100
• cognoms
• "Sr." + nom + " " + cognoms

Llocs on pot aparéixer una expressió

• **En una assignació:**
• area = base * altura / 2
• total += preu * quantitat
• **En els paràmetres d'una funció:**
• sqrt(b ** 2 - 4 * a * c)
• print("Sr." + nom + " " + cognoms)
• **En una condició:**
• if(edat < 0):
• while (nota < 0 or nota > 10)

El tipus de les expressions

Igual que una variable (o una constant) és d'un tipus determinat, una expressió també té el seu tipus.

Per exemple, tenim les variables enteres `a` i `b` i la variable *float* `x`.

- És lògic pensar que `a * b` també serà entera, i també `a + b`, etc

Ara bé:

- De quin tipus serà una expressió amb operands de diferents tipus: `a * x`?
- De quin tipus serà `a / b`? Enter (sense decimals) o *float* (amb decimals)?

En eixos casos cada llenguatge de programació fa una conversió de tipus o *promoció*. Hi ha diferents formes de *promoció*:

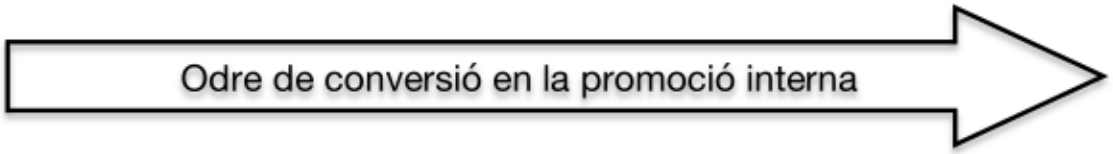
Promoció interna

Si en una expressió hi ha dades amb decimals i sense, **el resultat també tindrà decimals**.

Per tant, si tenim `4 + 2.3` el resultat serà `6.3` (*float*), no `6` (*int*).

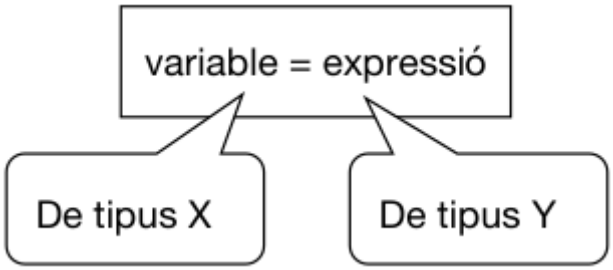
Esta norma serveix per a tots els llenguatges. Però C i Java tenen més tipus per a representar els números. En estos llenguatges, si una expressió té diferents tipus, **l'expressió serà del tipus que ocupa més bytes** (però si l'expressió té números decimals, l'expressió serà d'un tipus amb decimals).

PROMOCIÓ INTERNA EN C I JAVA								
	Números enters					Números amb decimals		
Tipus	char	int	unsigned	long	unsigned long	float	double	long double
Bytes	1	4	4	8	8	4	8	16



Promoció per assignació

Esta conversió la fa el compilador quan s'intenta assignar a una variable una expressió de diferent tipus.




```
preu = 3    # Ara preu és int (i el seu valor és 3)
preu = 4.6  # Ara preu és float (i el seu valor és 4.6)
```

```
int preu;   // Ara (i en tot el programa) preu serà int
preu = 3;   // preu ara val 3
preu = 4.6; // preu ara val 4 i continua sent int (en Python seria 4.6) (1)
```

1. En este cas diem que s'ha fet una promoció per assignació: el *float* 4.6 ha "promocionat" a l'*int* 4.

Promoció forçada (*càsting*)

El programador pot indicar que una expressió canvie a un tipus en concret. Eixa conversió es diu *càsting* o *promoció forçada*.

	Python	C i Java
Sintaxi	tipus(expressió)	(tipus) expressió
Exemples	float(x) int(euros * 166.386) str(961702294)	(float) x (int) (euros * 166.386) 961702294 + ""

O bé:
String.valueOf(961702294)

```
x = 4.6
n = int(x) * 2  # int(4.6) * 2 --> 4 * 2    --> 8 --> n valdrà 8
n = int(x * 2)  # int(4.6 * 2) --> int(9.2) --> 9 --> n valdrà 9

euros = 10
print(int(euros * 166.386)) # Mostrarà 1663 (sense deimals)

tel = 962829995 # tel és un enter i val 962829995. Podriem sumar-li números, etc.
tel = str(tel)  # tel ara és una cadena i val "962829995". Podriem concatenar...
```

Els tipus possibles en Python són: *int*, *float*, *str* i *bool*.

```
int n;
float x = 4.6;
n = ((int)x)*2;     // (4*2)        --> n valdrà 8
n = ((int)(x*2));   // (int)(9.2)   --> n valdrà 9

int n = 10;
int m = 3;
float x = 4.6;
x = (float)(n/m);   // (float)3         --> x valdrà 3.0
x = (float)n/m;     // (float)(10)/3    --> 10.0/3  --> x valdrà 3.333...
```

En este últim cas veiem que el càsting és útil per si volem fer una divisió d'emters però amb decimals en C i Java.

3.5 Precedència i associativitat d'operadors

Ací tenim una relació dels operadors ordenats per **procedència** de major a menor. En cas d'igualtat de precedència de divers operadors en una expressió, l'**associativitat** ens diu per on es comença a avaluar (d'esquerra a dreta o de dreta a esquerra).

CATEGORÍA DE L'OPERADOR	OPERADORS PYTHON	OPERADORS JAVA	ASSOCIATIVITAT
Parèntesi, vectors	( ) [ ]	( ) [ ]	ESQUERRA
Operadors unaris	+ -	++ -- + -	DRETA
Potència	**		DRETA
Multiplicació, divisió i residu	* / // %	* / %	ESQUERRA
Suma i resta	+ -	+ -	ESQUERRA
Operadors relacionals	< <= > >= == != <>	< <= > >= == !=	ESQUERRA
'No' lògic	not	!	ESQUERRA
'i' lògic	and	&&	ESQUERRA
'o' lògic	or	||	ESQUERRA
Operador condicional		?:	DRETA
Assignacions	= += -= = *= /= //= %=	= += -= *= /= %=	DRETA

No cal saber-se tot això de memòria. Simplement hem de saber que, per a avaluar les expressions, hi ha unes regles per vore què s'avalua primer. Davant del dubte, farem ús dels **parèntesis** per a indicar quines operacions volem que es facen primer.

Exercici 5. Expressions i tipus

Suposem que tenim estes variables en un programa en Python:

- `a = 12`
- `x = 2.5`
- `y = 0.6`

Indica de quin és el valor de cadascuna de les següents expressions:

- `a + x`
- `x + y`
- `int(x) + y`
- `int(x) + int(y)`
- `int(x + y)`
- `a / 4`
- `a // 4`
- `a % 4`
- `a + x * 2`
- `a / a - 2`
- `a ** 2 + 1`
- `a < x or y < x`
- `not(a < x)`
- `(a >= x) and (y <= a)`

3.6 Eixida de dades: *print*

Les instruccions d'entrada i eixida permeten la construcció de programes interactius. És a dir, amb elles podrem mostrar dades en pantalla i introduir dades per teclat.

En Python:

- Eixida de dades: *print*
- Entrada de dades: *input*

Exemple senzill de *print*

Esta funció mostra per pantalla allò que se li passa com a paràmetre. Ja ha aparegut anteriorment. Anem a detallar el seu funcionament.

Veiem estos exemples:

```
nom = "Pep"
cognoms= "Garcia"
print("Hola,", nom, cognoms)    # Mostra: Hola, Pep Garcia
print("Hola," + nom + cognoms)  # Mostra: Hola,PepGarcia
```

Com podem vore:

- *print* pot rebre una o més dades com a arguments (separats per comes).
- Si els arguments són textos han d'anar entre cometes (simples o dobles).
- Es mostren els textos separats per 1 espai. Al final es posa un salt de línia.
- Les cadenes de text poden concatenar-se amb l'operador '+' però no posa espais entre elles. I no podem concatenar un text amb un número. Donaria error.

Altres paràmetres del *print*

```
print(objecte/s, sep=separador, end=finalitzador, file=fitxer)
```

- **objecte/s** --> textos, números, variables o expressions que volem mostrar (separats er comes).

```
dies = 3
print("En", dies, "dies hi ha", dies*24, "hores")
```

```
En 3 dies hi ha 72 hores
```

- **sep=separador** --> Ací indicarem amb una cadena de caràcters com volem que apareguen separats els objectes que mostrem. Si no posem res, el separador és un espai en blanc.

```
print("LÍNIA", "ARTÍCLE ", "QUANT.", "PREU", "IMPORT", sep="\t")
print("------- --------------- ------- ------- ------")
print(1, "Tomaques", 3, 2.5, 3*2.5, sep="\t")
print(2, "Peres   ", 12.5, 1.5, 12*1.5, sep="\t")
print(3, "Plàtans ", 2, 2, 2*2, sep="\t")
```

```
LÍNIA   ARTÍCLE          QUANT.  PREU    IMPORT
------- --------------- ------- ------- ------
1       Tomaques        3       2.5     7.5
2       Peres           12.5    1.5     18.0
3       Plàtans         2       2       4
```

• **end=finalitzador** --> Cadena que es mostrarà al final del text. Si no posem res, el finalitzador és el fi de línia (caràcter '\n').

```
for n in range(1, 5): #(1)!
    print(n, end=" ") # Amb eixe end aconseguim que print NO faça intro sinó espai.
print() # En acabar fem un print() per a fer un únic intro al final.
```

1. Bucle on **n** va d'1 a 4 (ja ho veurem)

```
1 2 3 4
```

• **file=fitxer** --> Si no volem que es mostre per pantalla sinó a un fitxer.

```
fitxer=open("nomFitxer.txt", "a")   #(1)!
print("Hola, Pep", file=fitxer)
fitxer.close() #(2)!
```

1. El mode "a" afig el text al final del fitxer. El mode "w" reescriu el fitxer.
2. Fins que no es "tanca" el fitxer, no s'escriu el *buffer* en ell.

Usant el *print* amb format (ús de *f-strings*)

Suposem que volem mostrar els articles comprats, en forma de taula:

```
LIN ARTICLE                KILOS    PREU/KG IMPORT
--- -------------------- -------- -------- ------
  1 Tomaques del Mareny   3.0000     2.50   7.50
  2 Safrà iraní           0.0004 4266.67   1.71
```

Què passaria si cada part de cada línia la separem amb un tabulador?

```
print("LIN", "ARTICLE            ", "KILOS", "PREU/KG", "IMPORT", sep="\t")
print("---", "--------------------", "-----", "-------", "------", sep="\t")

linia = 1
article = "Tomaques del Mareny"
quantitat = 3
preu = 2.5
print(linia, article, quantitat, preu, quantitat*preu, sep="\t")

linia = 2
article = "Safrà iraní"
quantitat = 0.0004
preu = 4266.67
print(linia, article, quantitat, preu, quantitat*preu, sep="\t")
```

Segons les dades que tinguen les variables, podria ser que ens desquadrara tot:

```
LIN     ARTICLE                 KILOS   PREU/KG IMPORT
---     --------------------    -----   ------- ------
1       Tomaques del Mareny     3       2.5     7.5
2       Safrà iraní     0.0004  4266.67 1.706668
```

Això és degut a que els noms dels articles són de distinta llargària i els números no sempre tenen els mateixos decimals, etc.

Per a evitar això (a partir de la versió 3.6 de Python) tenim els *f-strings*, és a dir: "cadenes amb format", on podrem especificar el format que tindrà cada variable que posem en la cadena. Recordem que volíem que l'aspecte fora este:

```
LIN ARTICLE                KILOS   PREU/KG IMPORT
--- -------------------- ------- ------- ------
  1 Tomaques del Mareny   3.0000    2.50   7.50
  2 Safrà iraní           0.0004 4266.67   1.71
```

Per tant, li haurem de dir que en cada línia d'articles:

- El número de línia ocupe 3 posicions
- El nom de l'article ocupe 20 posicions
- Els kilos ocupen 7 posicions (4 d'elles que siguen decimals)
- El preu que ocupe 7 posicions (2 d'elles que siguen decimals)
- L'import que ocupe 6 posicions (2 d'elles decimals)

I la forma de fer-ho amb el *f-string* és així:

```
print(f"{linia:3} {article:20} {quantitat:7.4f} {preu:7.2f} {quantitat*preu:6.2f}")
```

És a dir, abans de la cadena es posa una *f* (o *F*), i cada variable o expressió es posa dins les claus i especificant el format després dels dos punts. Exemples:

```
nom = "Pep"
edat = 30
pi = 3.14159265359
print(f"El meu nom és {nom} i tinc {edat} anys")
print(f"Tinc {edat:<10} anys")  # Posa el número a l'esquerra dins de 10 caràcters
print(f"Tinc {edat:>10} anys")  # Posa el número a la dreta dins de 10 caràcters
print(f"Tinc {edat:^10} anys")  # Posa el número centrat dins de 10 caràcters
print(f"Pi és {pi:.2f} aprox")  # Mostra 2 decimals
print(f"Pi és {pi:10.2f} aprox")    # Mostra 2 decimals, a la dreta dins de 10 car.
```

El resultat seria:

```
El meu nom és Pep i tinc 30 anys
Tinc 30         anys
Tinc         30 anys
Tinc     30     anys
Pi és 3.14 aprox
Pi és       3.14 aprox
```

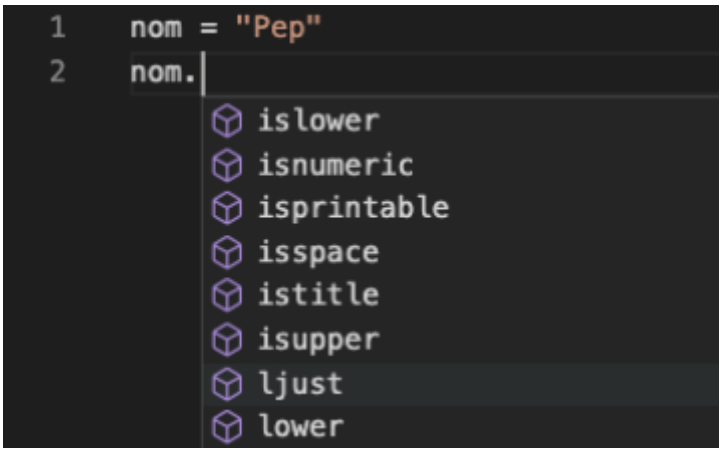
Observem que emprem `<`, `>` i `^` per a posar el valor a l'esquerra, dreta o centrat. I per als *float* indiquem també quants decimals volem (i li posem una `f`).

Realment els *f-strings* no tenen res a vore amb el *print*, sinó que serveix per a crear una cadena amb dades amb el format que volem. Per exemple:

```
base = float(input("Base: "))
altura = float(input("Altura: "))

area = (base * altura) / 2

missatge = f"L'àrea del triangle de base {base} i altura {altura} és {area:.2f}"

print(missatge)
```

A més, Python disposa de moltes **operacions que podem fer en una cadena**: passar-la a minúscules, a majúscules, obtindre una subcadena... I algunes de les operacions són alineació a dreta, esquerra, centrat.

En *VSCode* (i altre IDEs), si posem el punt al costat d'una variable (o constant) de tipus *str*, vorem les operacions que podem fer amb ella:



Per tant, podem fer altres coses en les cadenes sense usar els *f-strings*. Ací tenim uns exemples:

```
nom = "Pep"
print("Hola", nom.upper(), "com va?")          # Hola PEP com va?
print("Hola", nom.ljust(10, '_'), "com va?")   # Hola Pep_______ com va?
print("Hola", nom.center(10, '_'), "com va?")  # Hola ___Pep____ com va?
print("Hola", nom.rjust(10, '_'), "com va?")   # Hola _______Pep com va? (1)
```

1. En compte d'eixe '_' podem posar qualsevol caràcter, com per exemple un espai.

3.7 Entrada de dades: *input*

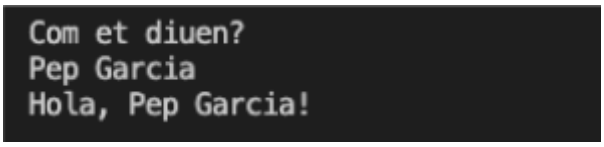
Serveix per a que un programa puga demanar dades per teclat. Serà un poc diferent segons els tipus de dades que volem introduir.

Vegem-ho amb exemples:

Entrada de text

```
print("Com et diuen?")
nom = input() #(1)!
print("Hola, " + nom + "!")
```

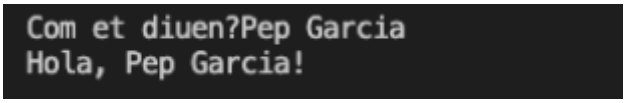
1. Ací l'execució espera que li posem per teclat un valor i polsem "INTRO". El valor introduït es guardarà a la variable *nom*.



Ara bé, l'*input* de Python també permet indicar el que estem demanant, sense haver de fer abans el *print*:

```
nom = input("Com et diuen?") #(1)!
print("Hola, " + nom + "!")
```

1. Ací mostrarà el text, i farà l'*input* normal.



Entrada de números

El problema és si, en compte de demanar un text per teclat, volem demanar un número, ja que l'agafarà com a text i no podrem fer operacions aritmètiques amb ell:

```
num = input("Dis-me un número: ")
print("El següent número és el ", num + 1)  # ERROR!
```

Això provoca l'error:

```
TypeError: can only concatenate str (not "int") to str
```

Això és degut a que *input* sempre retorna un *str*. Per això, en l'expressió *num + 1* intenta concatenar en compte de sumar. I dona error perquè no es poden concatenar números sinó textos.

Per tant, si volem tractar-lo com a enter caldrà fer un *càsting* (conversió de tipus):

```
num = input("Dis-me un número: ")
num = int(num)
print("El següent número és el ", num + 1)
```

```
Dis-me un número: 99
El següent número és el  101
```

O bé es podria fer l'*input* i el *càsting* en la mateixa instrucció:

```
num = int(input("Dis-me un número: "))  #(1)!
print("El següent número és el ", num + 1)
```

1. Es fa *càsting* sobre l'entrada de dades

En compte d'*int* també es pot fer casting a *float*, si fora el cas.

Diverses entrades en un mateix *input*

En un *input* podem demanar diverses dades separades per un espai en blanc (o pel caràcter que volem). Ara bé: això no té res a vore amb l'*input*, sinó amb el mètode ***split*** del tipus de dades *str*. Veiem uns exemples:

```
horaCompleta = input("Dis-me quina hora és (en format h:m:s): ")
hores, minuts, segons = horaCompleta.split(":") #(1)!

pes, altura = input("Dis-me el pes i altura (seprats per blanc): ").split()
```

1. Fem 3 assignacions alhora. *split* separa una cadena en una llista de cadenes.

```
Dis-me quina hora és (en format hh:mm:ss): 19:26:04
Dis-me el pes i altura (separats per blanc): 74 1.79
```

Veiem que quan fem *split*, en l'assignació cal posar tantes variables com dades s'espera que s'introduïsquen. Si no, donarà error.

Cal tindre en compte que després caldria fer els càstings corresponents a *int* o *float* de cada variable.

Exercici 6. Entrada i eixida de dades

Fes un programa que pregunte quants anys té algú i que mostre per pantalla la quantitat d'anys que falten per a la majoria d'edat i per a jubilar-se.

Exercici 7

Programa que pregunte per la base i l'altura d'un triangle i mostre per pantalla l'àrea d'eixe triangle

Exercici 8

Demana per teclat les dades de 2 llibres: títol, autor i preu (permet decimals). Després cal mostrar les dades en forma de taula: 30 caràcters per al títol, 20 per a l'autor i 10 per al preu (incloent 2 decimals i alineat a la dreta)

Exercici 9

Demana per teclat només un valor: una data (per exemple 6/9/2024). Després escriu eixa data però amb el format: "6 del 9 de 2024".

3.8 Exercicis

Exercici 10

Escriu el resultat de les següents expressions:

- `5 / 2 + 17 % 3`
- `3 * 6 / 2 + 18 / 3 * 2`
- `42 * 2 / 3 / (5 + 2)`
- `((5 + 3) / 2 * 3) / 2 - int(28.7) // 4 + 29 % 3 * 4`
- `3 <= 4`
- `45 <= 7 or not (5 >= 7)`
- `(8 * 2 < 5 or 7 + 2 > 9) and 8 - 5 < 18`
- `(2 * 7 > 5 or 7 / 2 == 3) and (7 > 25 or not True) and True`
- `35 > 47 and 9 == 9 or 35 != 3 + 2 and 3 >= 3`
- `9 == 15 or 8 != 5 and 7 == 4`
- `8 > 8 or 7 == 7 and not(5 < 5)`
- `4 + 2 < 8 and 24 + 1 == 25 or True`

Exercici 11

Escriu una expressió on s'especifique que una variable numèrica de nom ***quant*** siga menor o igual que 500 i múltiple de 5 però distinta de 100.

Exercici 12

Troba els errors en el següent programa que calcula l'àrea d'un cercle a partir del radi. Després copia'l amb les correccions i executa'l.

```
print("pi=", pi)
pi = 3,14
print(Programa de càlcul de l'àrea d'un cercle)
radi = input('Dis-me el radi');
'''Calcular i imprimir l'area
area = PI * radio**2;
print('\n\nL'àrea del cercle és: {aera:5.2}\n');
```

Exercici 13

Sense executar el programa, digues què mostrarà per pantalla:

```
a = 10
b = 3
c = a/b
d = a<b and b<c
a += a + b
b = float(a//b)
print(a, b, c, d)
```

Exercici 14

Contesta les següents qüestions tipus test:

1. Els tipus primitius en Python són:

• bool, char, short, int, long, float, double

• int, float, boo, str

• caràcters, variables i constants

2. Es defineix `a = 5`, `b = 2` i `c = 0`. Quin serà el valor de `c` després d'esta instrucció: `c = a > b`

• 3

• 2

• True

• False

• Error

3. Quin és el valor d'esta expressió: `10 / int(4.5)`

• 2

• 2.5

• 3

• Altra cosa

4. UD4. Estructures de control en Python

4.1 Sentències

Les sentències són cadascuna de les ordres que es donen a un programa. Hi ha de diferent tipus, independentment del llenguatge de programació:

- **D'importació**: Indiquen quines llibreries del programa van a fer-se servir. Com ja hem vist, les llibreries són un conjunt de funcions que ja estan fetes. Les sentències d'importació van al principi del programa.

```
import math

print("L'arrel de 25 és", math.sqrt(25))
```

- **Declaratives**: No és el cas de Python, però en C i Java vorem que per a emprar una variable primer cal declarar-la o definir-la (indicar el tipus). Això es fa en sentències declaratives. Ara bé, també entre en este grup les declaracions de funcions (ja entrarem en detall).

```
def saluda(nom):
    print("Hola, " + nom + "!")

saluda("Pep")
saluda("Maria")
```

- **Instruccions seqüencials**: Bàsicament són les instruccions d'assignació, de mostrar per pantalla i llegir de teclat. Encara que hi ha d'altres com: emetre un so, esborrar un fitxer... o invocar (cridar) a una funció.

```
edat = int(input("Edat: "))
any_naix = 2024 - edat
print("Nasqueres en", any_naix)
```

- **De control**: Són les que vorem en este tema. Permeten, sepenent d'una condició, executar unes instruccions o altres (**bifurcacions**) o repetir un conjunt d'instruccions (**bucles**).

```
edat = int(input("Edat: "))
if (edat >= 18):
    print("Vota")
else:
    print("No vota")
```

4.2 Bifurcacions

Les instruccions de bifurcació (o selecció) serveixen per a quan volem executar un conjunt d'rdres **només** si es compleix alguna condició determinada.

La instrucció que s'usa en tots els llenguatges és *if*, però cada llenguatge té la seua sintaxi. Veiem la de Python.

Bifurcació simple: *if*

S'executen unes instruccions només si es compleix una condició. Si no es compleix la condició no s'executa res en particular. Usarem:

```
edat = int(input("Quants anys tens? "))

if edat > 10:
    print("Com ja eres majoret puc dir-te que...")
    print("... els reis són els pares")

print("Bon Nadal")
```

Veiem que la sintaxi és:

```
if condició:
    acció_1
    acció_2
    ...
    acció_n
```

Bifurcació doble: *if-else*

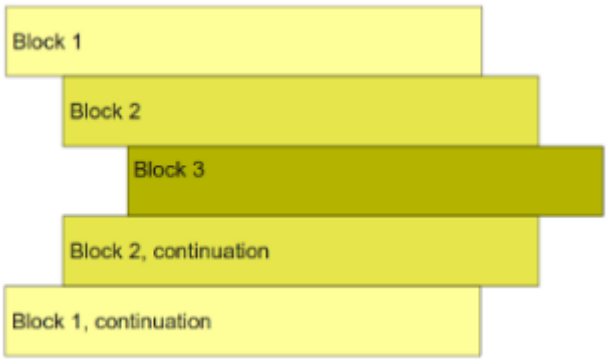
Si es compleix una condició s'executen unes instruccions. Si no es compleix, s'executen unes altres. Usarem *if* i *else*:

```
a = int(input("Dis-me un número: "))
b = int(input("Dis-me un altre número: "))

if a > b:
    major = a
else:
    major = b

print("El més gran és:", major)
```

Sagnat obligatori en Python

Les instruccions que depenen d'una condició es diu que formen un ***bloc***.

- Python: els blocs van **sagnats obligatòriament**, respecte la condició del bloc



- C, Java (i la majoria de llenguatges): els blocs van tancats entre claus { ... } i **preferiblement sagnats**.

```
...
if (condició) {
    ...
    ...
}
else {
    ...
    ...
}
```

Exercici 1. Bifurcacions

Fes un programa que calcule les solucions reals de l'equació de 2n grau `ax² + bx + c = 0` mitjançant la fórmula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

• S'ha de demanar per teclat quins valors tenen `a`, `b` i `c`.

• Cal comprovar si té solució (allò que està dins l'arrel ha de ser possitiu).

• Ha de mostrar les 2 solucions de la `x` (una amb el `+` i altra amb el `-`). O bé, només 1 solució si les 2 són la mateixa.

• Per calcular l'arrel usarem la funció `math.sqrt()`. de la llibreria *math*.

Bifurcació múltiple: *if-elif-...-else*

Podem pendre diferents accions en base a diferents condicions.

Emprarem 1 o diversos *elif*, que és una contracció d'*else if*.

```
estacio = int(input("Dis-me estació (1 a 4):"))

if estacio == 1:
    print("Primavera")
elif estacio == 2:
    print("Estiu")
elif estacio == 3:
    print("Tardor")
elif estació == 4:
    print("Hivern")
else:
    print("Error!")

print("Adéu")
```

Este és un exemple complet de sentència *if*. Com hem vist, les parts *elif* i *else* són opcionals. Per tant, la sintaxi completa de l'*if* és:

```
if condició_1:
    accions_1
elif condició_2:
    accions_2
...
elif condició_N:
    accions_N
else
    accionsElse
```

Exercici 2. Bifurcació múltiple

Demana una nota amb decimals i mostra el text corresponent: "ins", "suf", "bé", "not" o "exc". O bé "error" si la nota no està entre 0 i 10.

Exercici 3

Calculadora. Fes un programa que llija de teclat 2 números i una operació aritmètica (S/R/P/D). El prgrama farà el càlcul i imprimirà el resultat.

Altres consideracions sobre les bifurcacions

ifs niuats

Podem posar un *if* dins d'un altre, tant en la part de l'*if*, o d'algun *elif* o de l'*else*.

Exemple

```
edat = int(input("Quants anys tens? "))
if edat < 0:
    print("Error!")
else:
    print("Edat correcta")
    if edat < 12:
        print("No has fet ESO")
        if edat < 6:
            print("No has fet primària")
        else:
            print("Estàs en primària")
        print("Has de fer ESO")
    elif edat < 16:
        print("Estàs en ESO")
    elif edat < 65:
        print("Estàs en edat de treballar")
    else:
        print("Xe, jubila't!")
    print("T'he dit coses segons l'edat") #(1)!
print("Adéu")
```

1. Fixa't que els **sagnats** són molt importants. Estos 3 últims *print* s'executaran en situacions diferents.

Condicions compostes

Cada condició que es posa en els *if* (també en els *while*, com vorem després) pot ser composta. És a dir: diverses condicions unides amb ***or*** i/o ***and***, a més de poder tindre l'operador ***not***.

```
if edat >= 14 and edat <= 30:
    print("Pots traure't el Carnet Jove")
else:
    print("No pots traure't el Carnet Jove")
```

O bé, podem invertir els missatges si posem la condició contrària:

```
if not(edat >= 14 and edat <= 30):
    print("No pots traure't el Carnet Jove")
else:
    print("Pots traure't el Carnet Jove")
```

O bé, aplicant *De Morgan*:

```
if edat < 14 or edat > 30:
    print("No pots traure't el Carnet Jove")
else:
    print("Pots traure't el Carnet Jove")
```

Exercici 5. Bifurcacions amb condicions compostes

Encara que es pot fer amb condicions simples, fes-ho amb condicions compostes:

- Fes un programa que mostre el màxim de 3 números instroduïts per teclat.

Exercici 6

Encara que es pot fer amb condicions simples, fes-ho amb condicions compostes:

- Demana una nota sense decimals i mostra el text corresponent: `ins`, `suf`, `bé`, `not` o `exc`. O bé `ERROR` si la nota no està entre 0 i 10.

Exercici 7

Encara que es pot fer amb condicions simples, fes-ho amb condicions compostes:

- Demana per teclat el pes d'una persona, en quilos. Tant si el pes és menor de 10kg, com si és major de 200kg, ha de mostrar `ERROR`. En cas contrari, mostrarà el pes en grams i també un missatge: `flac` si és menor de 50kg, `normal` si està entre 50 i 100kg, o `sobrepés` en altre cas.

4.3 Bucles

A voltes necessitarem que un conjunt d'instruccions s'executen diverses vegades. Quantes vegades? Potser se sàpiga a priori o potser no. Per tant, tindrem dos tipus de bucles:

- **Bucles condicionals**: es repetirà mentre es complisca una condició.
- **Bucles incondicionals**: es repetirà un nombre determinat de vegades.

Bucles condicionals: *while*

Suposem que volem mostrar l'arrel quadrada d'un número introduït per teclat:

```
import math
num = int(input("Dis-me un número: "))
print("L'arrel quadrada de %d és %f" %(num, math.sqrt(num)))
```

Funcionarà bé a no ser que introduïm un número negatiu (ja que l'arrel quadrada d'un número negatiu no té solució real): *ValueError: math domain error*

Podríem solucionar-ho amb un *if*:

```
import math
num = int(input("Dis-me un número: "))

if num < 0:
    print("No es pot calcular l'arrel quadrada d'un número negatiu")
    num = int(input("Dis-me un número: "))

print("L'arrel quadrada de %d és %f" %(num, math.sqrt(num)))
```

Funcionarà bé si en el segon intent donem un número possitiu. Però i si tornem a donar un número negatiu en el segon intent? Quants *ifs* hauríem de posar? No ho sabem. Per tant, voldrem repetir el fet de demanar un número **MENTRE** el número introduït siga incorrecte:

```
import math
num = int(input("Dis-me un número: "))

while num < 0:
    print("No es pot calcular l'arrel quadrada d'un número negatiu")
    num = int(input("Dis-me un número: "))

print("L'arrel quadrada de %d és %f" %(num, math.sqrt(num)))
```

És a dir, **mentre** es complisca la **condició** de si el valor *num* és negatiu, s'executaran les instruccions de dins del bucle (les sagnades) i es tornarà a comprovar si *num* és negatiu. Quan ja no siga negatiu, eixirà del bucle i el programa continuarà fent les instruccions de fora del bucle.

A la vista d'este exemple veiem que:

- Usarem un bucle condicional *while* quan el programador **no sap quantes** vegades s'ahurà de repetir un conjunt d'instruccions.
- La variable (o variables) de la condició hauran de tindre **un valor abans** d'entrar al bucle.
- Dins del bucle ha d'haver alguna instrucció que permeta **canviar el valor** d'eixa variable (o variables) de la condició.

Sintaxi del *while* en Python:

```
inicialització de variables de la condició

while condició: # <--
    acció_1     #    ^
    acció_2     #    |
    ...         #    |
    acció_N     # -->

accions fora de bucle
```

És a dir: quan el flux del programa arriba al *while*, es comprova si la condició s'avalua a *True* o a *False*.

- Si és *True* s'executaran les accions 1 a N i el flux tornarà al *while*.
- Si és *False* el flux eixirà del bucle i s'executaran les accions de fora.

Exercici 8: Bucles condicionals

Fes un programa per controlar la temperatura d'un lloc que, **repetidament**, mostre un menú amb 4 opcions (Demanar temperatura / Pujar 1 grau / Baixar un grau / Eixir), que demane per teclat una opció i l'execute. Cada vegada que s'augmente o es disminuïsca, també es mostrarà la nova temperatura. Després del bucle es mostrarà quantes vegades s'ha canviat la temperatura.

Bucles incondicionals: *for*

Este tipus de bucles es fa quan el programadr ja sap **quantes vegades** s'han de repetir un conjunt d'instruccions (o bé eixa quantitat està guardada en una variable)

Exemples

- Si volem mostrar una taula de multiplicar, sabem que s'haurà de fer un print **10 vegades**.
- Si es demana per teclat de quants alumnes volem demanar l'edat, si eixe valor està a la variable *q_alumnes*, el bucle de demanar edats s'haurà de repetir ***q_alumnes vegades***.

En estos bucles hem d'anar comptant les iteracions fins arribar a la quantitat que volem. Per a això ens farà falta una variable, que farà de comptador o índex.

El bucle *for* en Python és bastant diferent al de C i Java.

Mostrar els números de l'1 al 9 (un en cada línia)

Python Java

```python
for i in range (1, 10):
    print(i)
```

```java
int i;
for(i=1; i<10; i++){
    System.out.println(i);
}
```

Anem a veure detalladament el *for* de Python. Al tema següent ja explicarem el *for* de Java.

Bucle *for* en Python

Recordem l'exemple que acabem de veure:

```
for i in range (1, 10):
    print(i)
```

Ací la variable ***i*** anirà agafant tots els valors que hi ha en la llista de valors representats en *range(1, 10)*. Per a entendre-ho, veiem com dunciona *range*:

La funció *range* de Python

Amb el *range* obtenim un conjunt de valors anomenat *llista* (ja parlarem més endavant de les llistes).

La funció admet 1, 2 o 3 paràmetres. Veiem exemples:

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

Exemples de Python amb *range*

Exemple 1 Exemple 2 Exemple 3

```
for i in range (10, 31, 5):
    print(i)
```

Això mostra els números: 10, 15, 20, 25, 30 (del 10 al 30, de 5 en 5)

```
for i in range (10, 31):
    if i % 5 == 0:
        print(i)
```

Este trauria el mateix resultat que l'Exemple 1

```
for i in range (10, 5):
    print(i)
```

Açò no mostraria res, ja que no es pot anar del 10 al 5 incrementant 1. Caldria fer:

```
for i in range (10, 4, -1):
    print(i)
```

Este sí que mostra els números: 10, 9, 8, 7, 6, 5 (del 10 al 5, de -1 en -1).

Exemples de *for* de Python sense *range*

Per a cada passada del *for* s'gafarà un element de la **lista** especificada:

Exemple 1 Exemple 2

```python
for num in [3, 5, 7, 11, 13, 17]:
    print(num, "és un número primer")
```

```
3 és un número primer
5 és un número primer
7 és un número primer
11 és un número primer
13 és un número primer
17 és un número primer
```

```python
alumnes = ["Pep", "Pepa", "Pepet"]

for nom in alumnes:
    print("Benvingut,", nom)
```

```
Benvingut, Pep
Benvingut, Pepa
Benvingut, Pepet
```

Exercici 9 RESOLT

Mostra els números del 0 al 10.

Python Java

```python
for i in range(11):
    print(i)
```

```java
for(int i=0; i<=10; i++){ //(1)!
    System.out.println(i);
}
```

1. No es precupeu ara per els *fors* de Java. Ja ho veurem al tema següent.

Exercici 10 RESOLT

Mostra la taula de multiplicar del 3.

Python Java

```python
for i in range(1, 11):
    print("3 x", i, "=", 3*i)
```

```java
for(int i=1; i<=10; i++){
    System.out.println("3 x" + i + " = " + (3*i));
}
```

Exercici 11 RESOLT

Mostra els números parells del 40 fins al 0 (inclòs).

Python Java

```python
for i in range(40, -1, -2): #(1)!
    print(i)
```

1. El rang de Python mai arriba al valor final. Per això no hem posat (40, **0**, -2), ja que així no haguera mostrat el 0.

```java
for(int i=40; i>=0; i-=2){
    System.out.println(i);
}
```

Exercici 12. Bucles amb *for*

Programa que demane una taula de multiplicar i la mostre.

Exercici 13

Programa que calcule el màxim de 10 números introduïts per teclat.

Exercici 14

Programa que calcule el màxim, mínim i mitjana de 10 números entrats per teclat.

Exercici 15

Programa que mostre les taules de multiplicar del 2 al 9.

Exercici 16

Programa que calcule el factorial d'un número introduït per teclat (n!) tenint en compte que:

• 0! = 1

• n! = n * (n-1) * (n-2) * ... * 2 * 1

Feu-ho amb diferents solucions:

• Amb un *for* recorrent els números des de l'1 fins n

• Amb un *for* recorrent els números des d'n fins a 1

• Amb un *while* recorrent els números des de l'1 fins n

• Amb un *while* recorrent els números des d'n fins a 1

Alteració de bucles: *break* i *continue*

Dins del bucle (tant *while* com *for*), podem fer que, si ocorre alguna determinada condició, alterar el funcionament normal del bucle. Això es pot fer amb les sentències *break* i *continue*:

```
...
# Inici del bucle ----------------------
for i in range (1, 10):
    ...
    if ... :
        continue #(1)!
    ...
    if ... :
        break #(2)!
    ...
# fi del bucle -------------------------
...
```

1. Si passa perl *continue*, el flux de control seguirà a l'inici del bucle. Si la *i* valia 7, ara valdrà 8.
2. Si passa pel *break*, el flux de control seguirà a la següent instrucció de fora del bucle.

És a dir:

- ***break*** interromp l'execució del bucle i seguim per la instrucció següent de fora del bucle.
- ***continue*** fa que el programa comence altra iteració, encara que no s'haja acabat l'actual. Per tant, les línies que hi ha dins d'un bucle a continuació del *continue* no s'executen en eixa iteració. Si el bucle és un *for*, anirem a l'increment e la variable comptadora o al següent element de la llista. Si és un *while*, es torna a comprovar la condició per a entrar altra vegada al bucle.

> **Nota**
>
> En cas de tindre estes instruccions dins de bucles niuats, nomñes afecten al bucle on estan firectament. Per exemple: si tenim un *break* en un bucle (pare) i este està dins d'un altre bucle (iaio), eixiríem del bucle pare però no del iaio.

Exercici 17 RESOLT. Bucles amb *break* i *continue*

Programa en Python que demane les notes dels alumnes (números enters i positius entre 0 i 10) i que després mostre quantes notes s'han introduït i la nota mitjana. Per a parar d'introduir notes caldrà posar la nota 11, que no es tindrà en compte per als càlculs.

NOTA: El programa es podria fer sense *break* ni *continue*, però ho farem així per a vore el seu funcionament.

```
# Inicialització de comptador i acumulador
notes = 0
suma = 0
# Demanar dades i fer càlculs
while True: #(1)!
    # Introducció de la nota per teclat
    nota = input("Dis-me una nota (11 per a ixir): ")
    # Comprovacions de nota OK
    if not nota.isnumeric(): #(2)!
        print("Han de ser números enters positius. Torna a provar")
        continue #(3)!
    nota = int(nota)
    if nota > 11:
        print("La nota màxima és 10. Torna a provar")
        continue #(4)!
    # Si volem acabar
    if nota == 11:
        print("Ja no et demane més notes. Vaig a eixir del bucle.")
        break #(5)!
    # Si tot ha anat bé, farem els càlculs
    print("Nota introduïda ok:", nota)
    notes += 1
    suma += nota
# Fi del bucle

# Mostrem els resultats
print(notes, "notes introduïdes.")
print("Nota mitja:", suma/notes)
```

1. Amb el *while True* aconseguim fer un bucle "infinit". Només podrem eixir si fem un *break* dins.
2. *isnumeric* és *True* si tots els caràcters són dígits.
3. Si passa per ací, torna altra vegada **a l'inici del bucle**.
4. Si passa per ací, torna altra vegada **a l'inici del bucle**.
5. Si passa per ací, eixim ja **fora del bucle**.

Exercici 18. Bucles amb *break* i *continue*

Programa que demane 10 números (positius i/o negatius) i mostre la mitjana només dels positius. Fes-ho a l'estil de l'exercici resolt anterior, per a fer ús del *continue*.

Ús de l'*else* de bucles amb *break*

Ja hem vist que l'*else* s'utilitza en sentències *if*. Però Python (no C i Java) permet posar-lo després d'un bucle en el qual s'haja fet algún *break*:

```
...
# Inici del bucle --------------------
while ... :
    ...
    if ... :
        break
    ...
# fi del bucle -----------------------
else: # Este else és del bucle, no de l'if
    ... # S'executaran estes accions si NO hem passat pel break.
...
```

La part de l'*else* s'executarà només quan el bucle ha acabat de manera *normal*:

- En un *while* quan ha acabat el bucle perquè no es complia la condició.
- En un *for* quan ha acabat el bucle perquè ja s'han recorregut els elements de la llista.

Exercici 19 RESOLT. Bucles amb *break* i *else*

Programa en Python que ens diga si ha trobat o no algun múltiple d'un número determinat en un cert rang de números.

```
inici = int(input("Des de quin número vols buscar múltiples? "))
limit = int(input("Fins quin número vols buscar múltiples? "))
divisor = int(input("De quin número vols búscar múltiples? "))

for i in range(inici, limit+1):
    print(i)
    if(i % divisor == 0):
        print("En el rang sí que hi ha almenys 1 múltiple de", divisor)
        break #(2)!
    else:
        print("Encara no he trobat múltiple de", divisor)

else: # else del bucle (1)
    print("En tot el rang no hi ha cap múltiple de", divisor)

print("Fi del programa")
```

1. Si el bucle acaba de forma *normal*, sí que s'executarà l'*else* de fora del bucle.
2. Si el bucle acaba per un *break*, no s'executarà l'*else* de fora del bucle.

Exercici 20

Fes un programa en Python que simule un caixer automàtic. Es demana per teclat la clau contínuament mentre no siga correcta (1234). Però com a molt són 5 intents. Després del bucle caldrà indicar si s'han superat els intents o si s'ha encertat la clau.

- Fes el programa usant un *while amb break* i l'*else* del bucle.
- Fes el programa usant un *for amb break* i l'*else* del bucle.
- Fes el programa usant un *while* sense *break* ni l'*else* del bucle.

4.4 La sentència *pass* de Python

La sentència *pass* no fa **res**.

Simplement està per a quan el programador vol deixar alguna part del programa temporalment sense indicar el codi, per alguna raó:

- Perquè està en les fases inicials del programa i ho deixa per a després o per a millorar la llegibilitat:

```
if nota >= 0 and nota <= 10:
    pass # He d'acabar el programa amb càlculs amb la nota (1)
else:
    print("Error en nota")
```

1. Si no es posa el *pass*, donaria error sintàctic.

```
if edat >= 18:
    pass # No fem res especial si té 18 anys o més
else:
    print("No pots votar")
```

4.5 Annex: Aleatoris amb Python

Per generar números aleatoris en Python, utilitzarem la llibreria *random*. Aquesta llibreria proporciona diverses funcions per generar números aleatoris de diferents tipus.

Per tant, haurem d'importar la llibreria *random* mitjançant l'expresió `import random` a l'inici del nostre programa.

Generar un número aleatori enter

Per generar un número enter aleatori dins d'un rang específic, utilitzem la funció `randint`:

```
import random

num = random.randint(1, 10)
print(num)
```

Això generarà un número enter aleatori entre 1 i 10 (ambdós inclosos).

Generar un número aleatori de coma flotant

Per generar un número de coma flotant aleatori entre 0.0 i 1.0, utilitzem la funció `random`:

```
import random

num = random.random()
print(num)
```

Per generar un número de coma flotant aleatori dins d'un rang específic, utilitzem la funció `uniform`:

```
import random

num = random.uniform(1.5, 5.5)
print(num)
```

Això generarà un número de coma flotant aleatori entre 1.5 i 5.5.

Seleccionar un element aleatori d'una llista

Per seleccionar un element aleatori d'una llista, utilitzem la funció `choice`:

```
import random

colors = ['roig', 'verd', 'blau', 'groc'] #(1)!
color_aleatori = random.choice(colors)
print(color_aleatori)
```

1. Malgrat que encara no hem vist l'us de llistes, aquest ús sembla prou intuitiu, no?

Amb aquestes funcions (i altres), podem generar números i seleccionar elements de manera aleatòria en Python. A continuació teniu l'enllaç a la documentació oficial:

Funció random

https://pgalera.github.io/prg/