Tema 3

Introducció a Python

0. Introducció

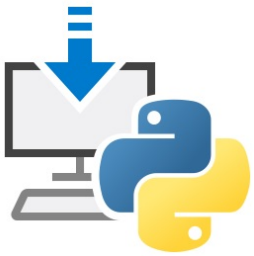
En este tema vorem una introducció al llenguatge de programació Python. Ara bé: com que més endavant es vorà el llenguatge Java, conforme anem veient coses de Python, anirem dient ja les diferències en Java. Estes diferències solen ser iguals en Java i en C.

1. Característiques bàsiques de Python

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

Python és un llenguatge de programació multi paradigma (programació imperativa, funcional, orientada a objectes, etc.), interpretat, de tipificació dinàmica i multi plataforma. Admet totes les estructures bàsiques de la programació, modularitat i és de propòsit general, fins i tot per a la creació de scripts.

1.1. Instal·lació de l'intèrpret de Python i d'un IDE de Python



Per a poder programar en Python cal instal·lar l'intèrpret de Python al nostre sistema. Això ho podem fer de moltes maneres, depenent de si ho instal·lem en una màquina virtual, en un contenidor *Docker* o directament (Linux ja el porta integrat). També existixen intèrprets online (recomanat), de manera que no cal instal·lar res. Un dels intèrprets online és *onlinegdb.com*.

En cas de voler-lo instal·lar al sistema, cal descarregar l'intèrpret de Python per al sistema operatiu on volem instal·lar-lo. *www.python.org/downloads/*

El que necessitem per a escriure un programa en Python és simplement un editor, com per exemple el bloc de notes. Després d'escriure'l el guardarem amb un nom (normalment amb extensió **.py*) i ja podem passar a executar-lo, ja que és un llenguatge interpretat i no cal compilar-lo.

Però no ho farem amb el bloc de notes sinó amb un IDE (Entorn Integrat de Desenvolupament). Usarem el *Visual Studio Code* amb els *plugins* (complements) necessaris. En este IDE, per a accedir al mode interactiu (o imperatiu) cal anar a la finestra de baix (on estan els símbols >>>) o bé seleccionar el tros de programa que volem executar i polsar *shift+enter*.

1.2. Estructura d'un programa en Python

Un programa en Python té una estructura molt simple (a diferència d'altres llenguatges):

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

Qualsevol programa escrit en Python té la següent estructura:

[Descripció del programa]

[Directives]

[Importació de llibreries]

[Definició de classes]

[Definició de funcions]

Entrada de dades

Processament de dades

Sortida de dades

Els claudàtors [...] indiquen que eixos apartats són opcionals.

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.

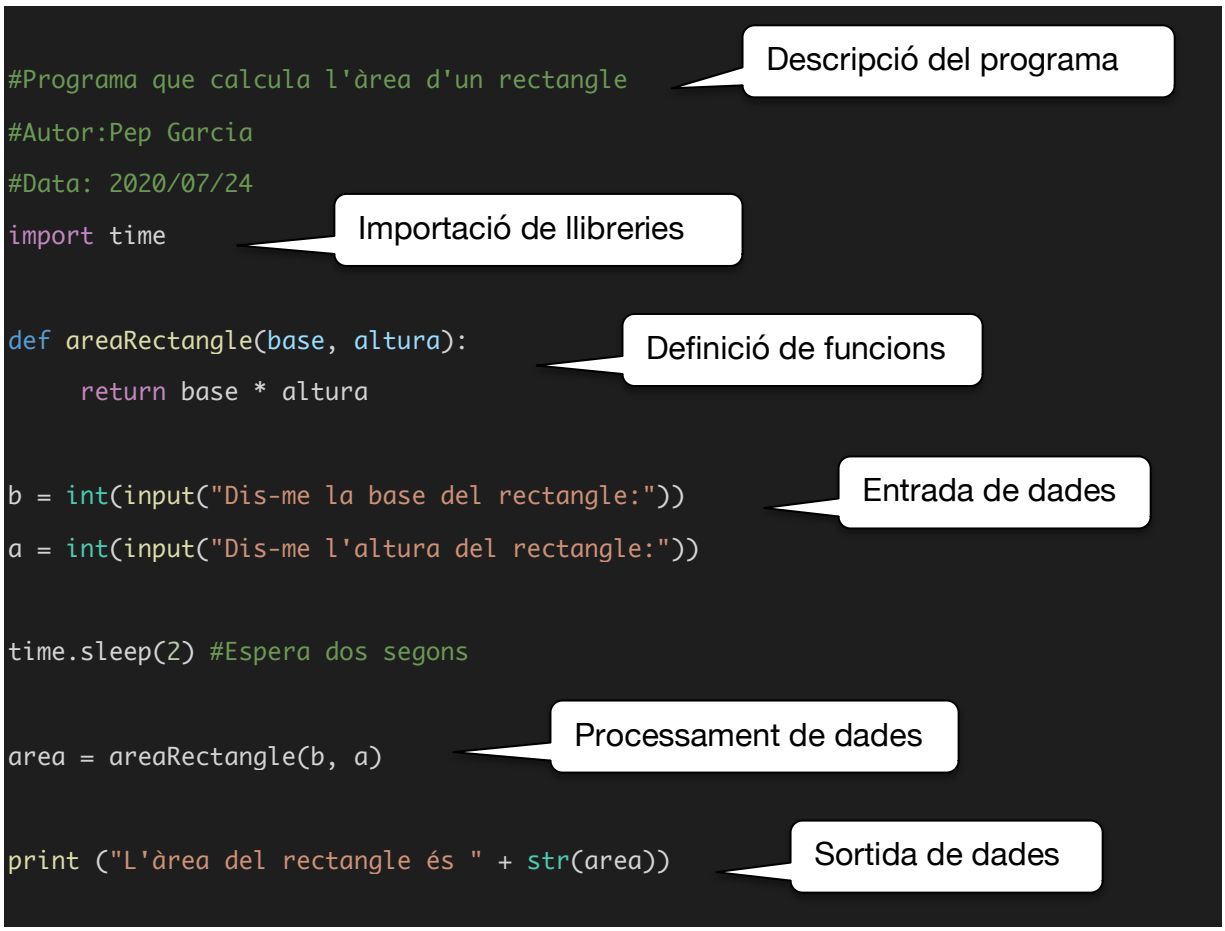
En Python estos apartats poden variar de posició.

Veiem uns exemples i els analitzem.

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

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

Un altre exemple, més complet:



Copia i apega este codi en l'IDE que tingues de Python i executa-ho per vore què fa.

Anem a explicar l'exemple un poc per damunt. Tranquils si no s'entén ara, ja que cada part s'explicarà més endavant amb detall.

- Descripció 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 programa 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 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'introduÏsquen dades per teclat.
    - Es fa la crida a la funció que hem definit abans (*areaRectangle*).
    - Finalment mostrem a l'usuari el resultat (*print*).

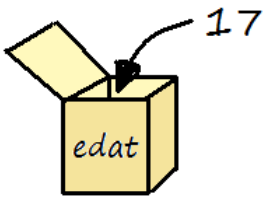
1.3. Noms de variables i funcions en Python

Abans hem vist que hem posat noms de variables 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:

- Són una combinació de lletres minúscules [a..z], majúscules [A..Z], dígits [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	

1.4. 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.

1.5. 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.

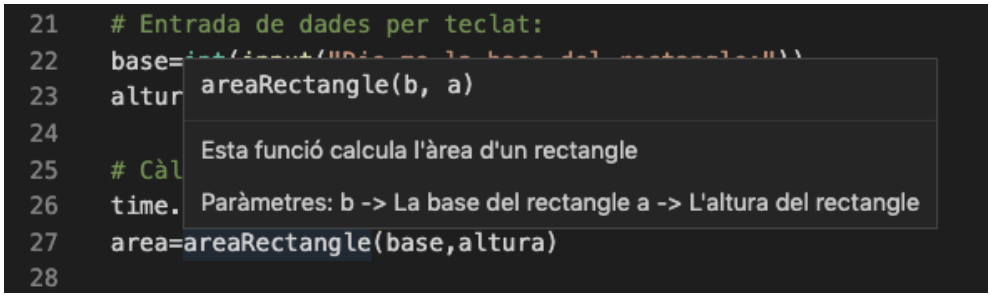
Tipus de comentaris:

- **D'una línia**:
  - Precedits 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, després podrem accedir a eixe comentari posant: *help*(nomFunció).

Veiem en este programa exemples dels diferents tipus de comentaris:

```
'''
Programa que calcula l'àrea d'un rectangle.
Autor: Pep Garcia
Data:26-8-2020
'''
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))
```

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



1.6. Delimitadors

Són 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 punt i coma (;) però Python fa servir simplement el bot de línia.

Ací tenim els delimitadors que s'usen en Python.

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

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.

2.1. 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 signe o sense... Igual que per a números amb decimals.

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

El *print* mostra el número en sistema decimal.

2.1.2. 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
```

2.1.3. Lògics: *bool*

Serveix per si una variable volem que tinga 2 únics estats (vertader o fals). 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*).

2.1.4. 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 Punta, 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 dins 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')
```

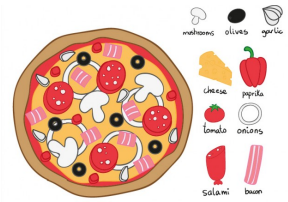
En una cadena de text també podem utilitzar este caràcter d'escapament per a representar diverses accions:

SEQÜÈNCIES D'ESCAPAMENT			
Python, C i Java	Acció	Exemple	Resultat
\n	Nova línia	print("Hola\nAdéu")	Hola Adéu
\t	Tabulador	print("Hola\tAdéu)	Hola Adéu
\r	Retorn de carro	print("Hola\rTu)	Tula
\b	Backspace	print("Hola\bAdéu)	HolAdéu

Encara que les més usades són \n i \t.

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

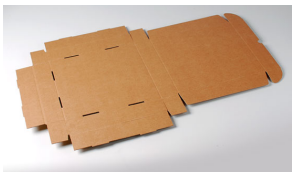
2.2. Tipus composts



Els tipus simples (que acabem de vore) serveixen quan hem que guardar una informació simple. És a dir, formada per una sola dada (una temperatura, 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.

2.3. 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:

2.3.1. Llenguatges de tipificació estàtica (C, Java...)

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.

Exemple C i Java:

```
int n;
float x;
n = 3;
x = 4.5;
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.
```

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

En estos llenguatges **no cal declarar 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.

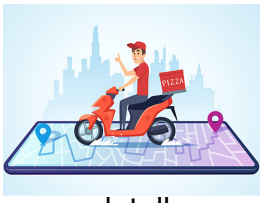
Exemples en Python:

```
n=7     # Primera vegada que ix la variable n. Ara la n val 7 Per tant, és int.
n=5.67  # Ara n val 5.67. Per tant, ara és float.
n=9     # Ara n val 9. Per tant, continua sent int.
n=n+2   # Ara n 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.

2.4. Àmbit i visibilitat



Nota: Estos 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 la funció "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: La variable 'a' és **global**

```
# ----------------------
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 esta "declarada" (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 la funció "funcioneta"
    - En l'exemple 2, l'àmbit de 'a' és 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 tros 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
else:
    major = False

print(major)
```

El primer ús de "major" es fa en este bloc

Però Python em permet usar-la fora del bloc

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

```
edat = False

if (edat >= 18):
    major = True

print(major)
```

Declarem la variable en el "bloc de fora"

... ja que vaig a usar-la en eixe bloc

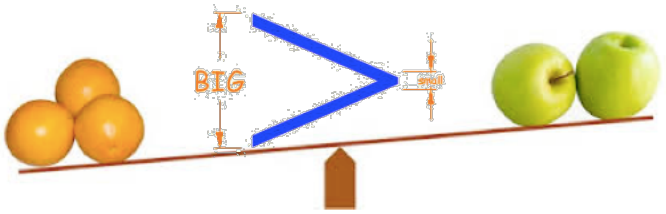
3. Operadors

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

3.1. Operadors aritmètics

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 de divisió entera 21 % 4 → 1	
**	NO	Potència 2 ** 3 → 8	

3.2. Operadors relacionals



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

Ja vorem que, principalment, estos operadors s'utilitzen en les condicions de les instruccions *if* i *while*. De moment, veiem uns exemples:

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

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

3.3. Operadors lògics

and, or, not

Ja veiérem al primer tema quins eren els operadors lògics i com actuaven (recordeu les "taules de veritat"). Veiem ara com es representen en ython, C i Java:

OPERADORS LÒGICS		
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 a control d'errors i a velocitat d'execució.

Exemples

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

Si el descompte_1 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

1. Què passaria en cada cas?

a)

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

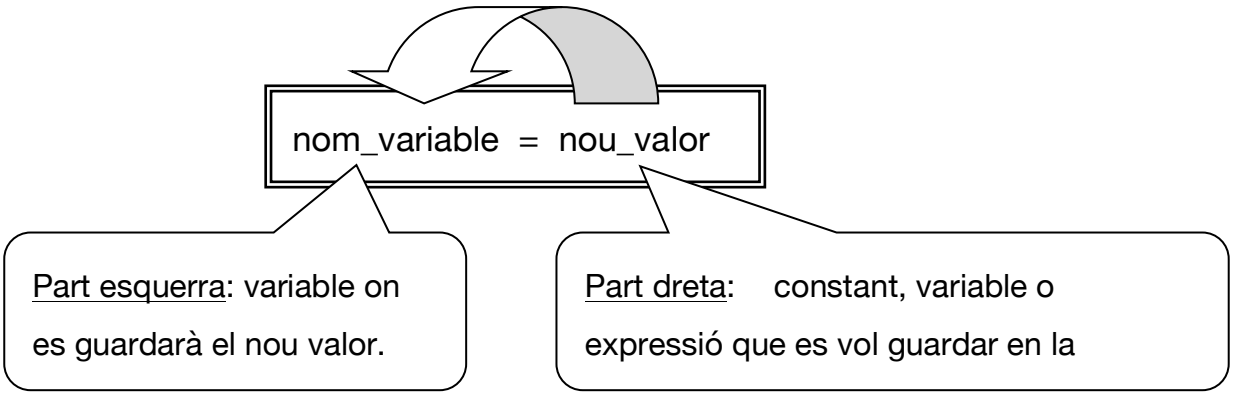
b) El mateix programa però amb este if en compte de l'altre:

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

3.4. Operador d'assignació

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

OPERADOR D'ASSIGNACIÓ		
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 de l'esquerra.

Exemple:

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

3.5. 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	++	Auto increment	x++	x = x + 1
NO	--	Auto decrement	x--	x = x - 1

Com veiem, estos 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
```

Exercicis sobre assignacions

2. En el 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 =
```

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 - 9    # a =       b =
```

3.6. 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 versió 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:

En Python: sys.getsizeof( )

```python
import sys
x = 10
text = "10"
print(sys.getsizeof(x), "bytes")            # 28 bytes (qualsevol enter ocupa 28 bytes)
print(sys.getsizeof(text), "bytes")         # 51 bytes (el text ocupa 49 bytes més la quantitat de lletres)
print(sys.getsizeof(1000), "bytes")         # 28 bytes
print(sys.getsizeof("Hola, Pep"), "bytes")  # 58 bytes
print(sys.getsizeof(1234.56789), "bytes")   # 24 bytes (un float ocupa 24 bytes)
```

En C (però no Java): sizeof( )

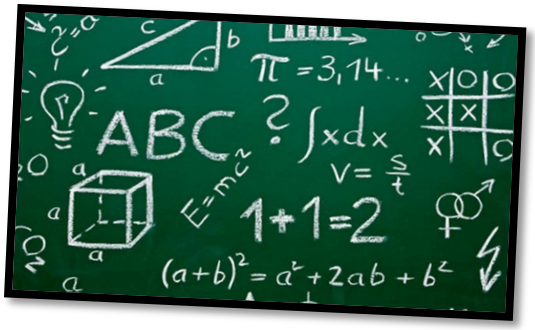
L'operador sizeof() calcula la quantitat de bytes que ocupa la variable o tipus que li posem dins dels parèntesis.

```c
int x = 10;
char text[] = "10";
printf("%lu bytes\n", sizeof(x));        // 4  bytes (qualsevol enter ocupa 4 bytes)
printf("%lu bytes\n", sizeof(text));     // 3  bytes (el text ocupa 1 byte més la quantitat de lletres)
printf("%lu bytes\n", sizeof(1000));     // 4  bytes
printf("%lu bytes\n", sizeof("Hola, Pep"));  // 10 bytes
printf("%lu bytes\n", sizeof(1234.56789f));  // 4  bytes (qualsevol float ocupa 4 bytes)
printf("%lu bytes\n", sizeof(1234.56789));   // 8  bytes (qualsevol double ocupa 8 bytes)
```

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 parèntesis per a indicar l'ordre de càlcul.



Exemples d'expressions:

- base * altura / 2
- (-b + sqrt(b**2 - 4*a*c))/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 ):
```

4.1. El tipus de les expressions

Igual que un 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":

4.1.1. Promoció interna

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

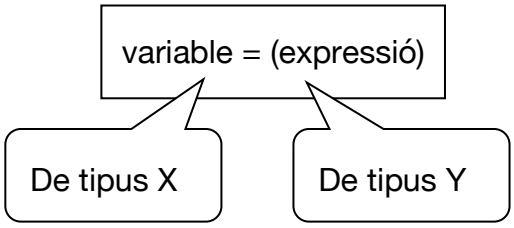
Per tant, si tenim 4 + 2.3 el resultat serà 6.3 (*float*), no 6 (*int*).

- 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

4.1.2. Promoció per assignació

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



- **Python**: la variable passarà a ser del tipus de l'expressió

```
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)
```

- **C, Java**: el tipus de la variable no canvia. És l'expressió la que canvia al tipus de la variable perquè es faça l'assignació.

```
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)
```

En este cas diem que s'ha fet una promoció per assignació.

4.1.3. 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	int(x) float(euros * 166.386) str(961702294)	(int) x (float) (euros * 166.386)

Exemples d'ús en Python:

```
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 decimals)

tel = 961702294     # tel és un enter i val 961702294
tel = (str) tel     # tel ara és una cadena i val "961702294"
```

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

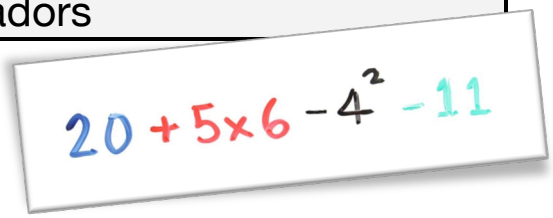
Exemples d'ús en C i Java:

```
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; m = 3;
float x;
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'enters però amb decimals en C i Java.

5. Precedència i associativitat d'operadors



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

CATEGORIA DE L'OPERADOR	OPERADORS PYTHON	OPERADORS JAVA	ASSOCIATIVITAT
Parèntesis, vectors	( ) [ ]	( ) [ ]	ESQUERRA
Operadors unaris	+ -	++ -- + -	DRETA
Potència	**	NO	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	NO	? :	DRETA
Assignacions	= += -= *= **= /= //= %=	= += -= *= /= %=	DRETA

Però 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.

Exercicis sobre expressions i tipus

5. 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) a + x

b) x + y

c) int(x) + y

d) int(x) + int(y)

e) int(x + y)

f) a / 4

g) a // 4

h) a % 4

i) a + x * 2

j) a / a - 2

k) a ** 2 + 1

l) a < x or y < x

m) not (a < x)

n) (a >= x) and (y<=a)

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*

6.1. Exemple senzill de *print*

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

6.2. Altres paràmetres del *print*

print(objecte/s, *sep*=separador, *end*=finalitzador, *file*=fitxer, *flush*=*True*/*False*)

- **objecte/s** → textos, números, variables o expressions que volem mostrar (separats per comes)
- ***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.
- ***end*=finalitzador** → Cadena que es mostrarà al final del text. Si no posem res, el finalitzador és el fi de línia (caràcter '\n')
- ***file*=fitxer** → Si no indiquem esta part, el text apareix en pantalla. Si en compte d'això volem que vaja a un fitxer de text, ho indicarem així: *file=open("nomFitxer.txt", "a+")*. Però ja ho vorem en detall a final de curs.
- ***flush=True/False*** → Suposem que al llarg d'un programa tenim molts prints. Si volguérem que tots es mostraren al final del programa (i no immediatament després d'executar-se cada instrucció), caldrà posar *flush=True*. Si no s'indica res, és *False* i, per tant, per defecte fa el funcionament normal. Esta opció no sol utilitzar-se massa.

Exemples:

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

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

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

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

Instrucció que repetix 4 vegades el *print*. Ja ho vorem.

```
for n in range(1,5):
    print(n)        # Fa un intro després de cada print
```

```
1
2
3
4
```

Però... i si no volem un intro després de cada número? Cal dir-ho amb l'*end*:

```
for n in range(1,5):
    print(n, end=" ") # Amb l'end aconseguim no fer intros
print()      # En acabar fem un print() per a fer un únic intro al final.
```

```
1 2 3 4
```

L'operador + concatena cadenes, no cadenes amb números. Si volem concatenar una cadena a un número, hem de fer el càsting d'eixe número a cadena (amb *str*):

```
anys = 20
print("Pep té", anys, "anys")
print("Pep té " + anys + " anys")   # Error: no es poden concatenar números
print("Pep té " + str(anys) + " anys") # Amb el càsting, sí.
```

6.3 Usant el *print* amb format

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

```
print("LIN", "ARTICLE", "KILOS", "PREU/KG", "IMPORT", sep="\t")
print("___", "_______", "_____", "_______", "______", sep="\t")
print("")
print(1, "Tomaques del Mareny", 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")
print(4, "Safrà iraní", 0.0004, 4266.67, 0.0004*4266.67, sep="\t" )
```

```
LIN     ARTICLE KILOS   PREU/KG IMPORT
___     _______ _____   _______ ______

1       Tomaques del Mareny     3       2.5     7.5
2       Peres   12.5    1.5     3.0
3       Plàtans 2       2       4
4       Safrà iraní     0.0004  4266.67 1.706668
```

Encara que usem tabuladors, les columnes desquadren ja que els noms dels articles són de distinta llargària i els números no sempre tenen els mateixos decimals, etc

Per a solucionar això hi ha una forma de dir en el *print* en quants espais es mostrarà un text o un número, així com quants decimals volem mostrar. És a dir: volem que es mostre així:

```
LIN       ARTICLE          KILOS  PREU/KG IMPORT
___ ____________________ _______ _______ ______

  1  Tomaques del Mareny   3.0000    2.50  7.50
  2  Peres                12.5000    1.50 18.00
  3  Plàtans               2.0000    2.00  4.00
  4  Safrà iraní           0.0004 4266.67  1.71
```

Caldrà dir-li que per a la descripció de l'article sempre reserve 20 posicions, per als quilos 7 posicions, 4 d'elles per als decimals, etc.

Ara vorem com fer-ho.

6.3.1. Especificació de format per a números enters

Exemple:

```
preu = 25
print("Val %4d euros" % preu)  # 4 espais per a posar el valor de la variable preu
```

```
Val   25 euros
```

És a dir:

- En la cadena de text es posa el format per al número amb el %d i, enmig, la quantitat de caràcters reservats per al número.
- Fora de la cadena de text es posa altre % i el valor o variable.

Altres exemples:

```
print("%2d multiplicat per %2d fan %3d" % (3, 2, 6))
print("%2d multiplicat per %2d fan %3d" % (10, 5, 50))
print("%2d multiplicat per %2d fan %3d" % (10, 10, 100))
```

```
 3 multiplicat per  2 fan   6
10 multiplicat per  5 fan  50
10 multiplicat per 10 fan 100
```

És a dir: si una cadena té diversos %, després es posen tots els valors corresponents en eixe ordre, tancats entre parèntesis i separats per comes.

6.3.2. Especificació de format per a números amb decimals

Podem usar el mateix sistema per a representar números amb decimals:

```
kilos = 2.376
print("Són %6.2f kg" % kilos)
```

```
Són   2.38 kg
```

- Usarem %f en compte de %d
- Entre % i la f posarem 2 valors: la quantitat total de caràcters reservats i la quantitat de números decimals que volem mostrar.

6.3.3. Especificació de format per a text

Seguint el mateix esquema, per a mostrar una variable (o constant) de text podem indicar en quants espais la volem representar:

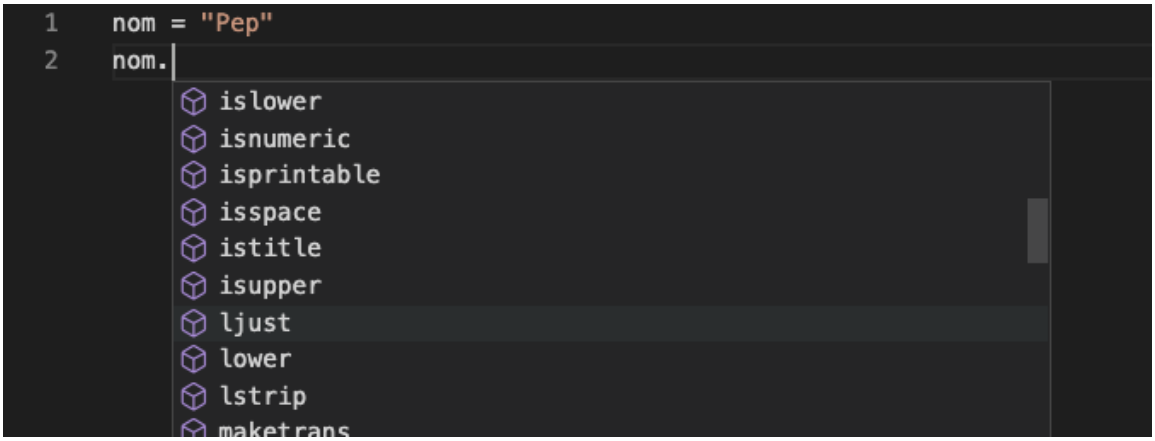
```
nom = "Pep"
print("Hola %s, com va?" % nom)
print("Hola %5s, com va?" % nom)
print("Hola %2s, com va?" % nom) # Si no cap el nom en 2 espais, posa el nom sencer
```

```
Hola Pep, com va?
Hola   Pep, com va?
Hola Pep, com va?
```

Veiem que l'especificador de format per a les cadenes és %s. Entre el % i la s podem posar la quantitat de caràcters que volem que ocupe la cadena.

Ara bé, independentment del *print*, 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.

Si posem el punt al costat d'una variable (o constant) de tipus *str*, vorem les operacions que podem fer amb ella:



Per tant, en compte d'indicar les posicions dins del %s, podem usar estes funcions que, a més, tenen més opcions de format:

```
print("Hola %s, com va?" % nom.ljust(10, '_'))
print("Hola %s, com va?" % nom.center(10, '_'))
print("Hola %s, com va?" % nom.rjust(10, '_'))
```

```
Hola Pep_______, com va?
Hola ___Pep____, com va
Hola _______Pep, com va?
```

Per tant, a partir del format de números i textos, podem fer coses com el que volíem fer:

```
LÍN      ARTICLE          KILOS  PREU/KG IMPORT
___ ____________________ _______ _______ ______

 1  Tomaques del Mareny   3.0000    2.50  7.50
 2  Peres                12.5000    1.50 18.00
 3  Plàtans               2.0000    2.00  4.00
 4  Safrà iraní           0.0004 4266.67  1.71
```

Així:

```
print("LÍN".center(3,' '), "ARTICLE".center(20,' '), "KILOS".center(7,' '),
"PREU/KG".center(6,' '), "IMPORT".center(6,' ') )
print("_".center(3,'_'), "_".center(20,'_'), "_".center(7,'_'), "_".center(7,'_'),
"_".center(6,'_') )
print("")
print("%2d "%(1), "Tomaques del Mareny".ljust(20, ' '), "%7.4f %7.2f %5.2f"%(3,
2.5, 3*2.5))
print("%2d "%(2), "Peres".ljust(20, ' '), "%7.4f %7.2f %5.2f"%(12.5, 1.5, 2*1.5))
print("%2d "%(3), "Plàtans".ljust(20, ' '), "%7.4f %7.2f %5.2f"%(2, 2, 2*2))
print("%2d "%(4), "Safrà iraní".ljust(20, ' '), "%7.4f %7.2f %5.2f"%(0.0004,
4266.67, 0.0004*4266.67))
```

7. Entrada de dades: *input*

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

Vegem-ho amb exemples:

7.1. Entrada de text

```
print("Com et diuen?")
nom = input()

print("Hola, " + nom +"!")
```

```
Com et diuen?
Pep Garcia
Hola, Pep Garcia!
```

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?")
print("Hola, " + nom +"!")
```

```
Com et diuen?Pep Garcia
Hola, Pep Garcia!
```

Veiem que, en este cas, no ha fet intro (salt de línia) després de preguntar "Com et diuen?". Caldria haver posat el \n:

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

```
Com et diuen?
Pep Garcia
Hola, Pep Garcia!
```

7.2. 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:

```
edat = input("Dis-me un número: ")
print("El següent número és el ", edat + 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 *edat + 1* intenta concatenar en compte de sumar. I dona error perquè no es poden concatenar números sinó textos.

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

```
edat = input("Dis-me un número: ")
edat = int(edat)    # Ací fem el càsting o conversió de tipus
print("El següent número és el ", edat + 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:

```
edat = int( input("Dis-me un número: ") )  # Es fa càsting sobre l'entrada de dades
print("El següent número és el ", edat + 1)
```

En compte d'*int* també es podria fer el càsting 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 hh:mm:ss): ")
hores, minuts, segons = horaCompleta.split(":")


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

```
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.

Exercicis sobre entrada i eixida de dades

6. 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.

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

8. Demana per teclat les dades de 3 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). Per exemple:

Diccionari per a ociosos	Joan Fuster	10.40
L'home manuscrit	Manuel Baixauli	14.25
Un nu	Josep Palàcios	9.50

9. Demana per teclat la data de hui (separat amb el caràcter /). Després escriu la data amb el format d'este exemple: "4 del 8 de 2020".

8. Exercicis

1. Escriu el resultat de les següents expressions:

    a) 5 / 2 + 17 % 3

    b) 3 * 6 / 2 + 18 / 3 * 2;

    c) 42* 2 / 3 / (5 + 2)

    d) ((5+3)/2*3)/2-int(28.7)//4+29%3*4

    e) 3 <= 4

    f) 45 <= 7 or not(5 >= 7)

    g) (8 * 2 < 5 or 7 + 2 > 9) and 8 – 5 < 18

    h) (2 * 7 > 5 or 7 / 2 == 3) and (7 > 25 or !True) and True

    i) 35 > 47 and 9 == 9 or 35 != 3 + 2 and 3 >= 3

    j) 9 == 15 or 8 != 5 and 7 == 4

    k) 8 > 8 or 7 = 7 and not(5 < 5)

    l) 4 + 2 < 8 and 24 + 1 == 25 or True

2. 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.

3. Troba els errors en el següent programa que calcula l'àrea d'un cercle a partir del radi. Després copia'l amb les correccions, compila'l 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'àrea
area = PI * radio**2;
print('\n\nL'àrea del cercle és: %.2f\n', aera);
```

4. 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)
```

5. Contesta les següents qüestions tipus test:

   a) Els tipus primitius en Python són:

      1. bool, char, short, int, long, float, double
      2. int, float, bool, str
      3. caràcters, variables i constants

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

      1. 3
      2. 2
      3. True
      4. False
      5. Error

   c) Quin és el valor d'esta expressió: 10 / int(4.5):

      1. 2
      2. 2.5
      3. 3
      4. Altra cosa