Iniciació a Python

Ajuntament de Barcelona

Barcelona Activa

Índex

1 QUÈ ÉS PYTHON I COM COMENÇAR A PROGRAMAR

Començarem descobrint els trets més característics de Python i veient les seves aplicacions més rellevants. Hi ha molts llenguatges de programació, però Python s'ha convertit en un dels més populars arreu.

Després de la introducció al llenguatge, passarem a instal·lar l'intèrpret, el responsable de què el nostre codi sigui entès per l'ordinador.

Més tard, instal·larem pas a pas l'entorn de programació que ens facilitarà la gestió del nostre codi.

Amb l'intèrpret i l'entorn de programació ja tindrem totes les eines per començar a programar. Només ens caldrà tenir ganes d'aprendre i, potser, en certs moments, una mica de paciència si no ens surten les coses a la primera. El món Python ens espera!

1.1 QUÈ ÉS PYTHON I QUINES APLICACIONS TÉ

Us donem la benvinguda al món Python!

T'agrada el desenvolupament web i vols crear aplicacions? Has pensat mai en dissenyar un videojoc? T'interessa el *big data* i la intel·ligència artificial?

Sí, tot això es pot fer amb Python. Si tens imaginació i ganes d'aprendre a programar amb aquest llenguatge, un dia podràs arribar a crear allò que imaginis o trobar aquella feina que tant havies desitjat.

Així que no perdem més temps i comencem a conèixer què es realment Python. Anem, doncs, a destacar les seves característiques principals que el fan tan popular:

- És un llenguatge de codi lliure. Pots modificar o millorar tot allò que vulguis del codi font.

- Està basat en l'orientació a objectes. El teu codi s'escriurà en blocs reaprofitables que tindran funcions i variables que podràs crear seguint el teu propi criteri, sempre i quan compleixis una sèrie de normes.

- Fàcil d'aprendre. Python s'escriu fent ús d'una sintaxi clara i neta que et permet centrar-te en el desenvolupament del codi. A més, disposa de moltes funcions i llibreries que podràs aprofitar, sense necessitat de construir-les des de zero.

- És un llenguatge interpretat i multiplataforma. L'intèrpret traduirà el teu codi sense necessitat de compilar-ho, per tal que l'ordinador el pugui executar a mesura que sigui necessari i, típicament, instrucció per instrucció. Així que, fent ús d'un intèrpret, el mateix fitxer, és a dir, el teu codi, podrà ser executat en sistemes molt diferents. Per això diem que també és un llenguatge multiplataforma.

Per ser membre rellevant de Python i escriure codi de qualitat, hauràs de tenir en compte aquests principis, que constitueixen *The Zen of Python.*

- Bonic és millor que lleig.
- Simple és millor que complex.
- És important que es llegeixi bé.
- Dispers és millor que dens.
- Els errors mai no s'han de silenciar.
- Davant de l'ambigüitat, rebutja endevinar.
- Ara és millor que mai… tot i que mai és normalment millor que ara mateix.
- Si la implementació és difícil d'explicar, deu ser una mala idea.

Potser aquests principis no et diuen res ara, però guarda'ls en lloc segur. Quan comencis a escriure codi, torna'ls a llegir. Et guiaran i t'inspiraran.

Escriure codi amb Python és molt divertit. En aquest curs, ràpidament començaràs a escriure els teus propis algoritmes. Programar et posarà davant de reptes emocionants i t'animarà a superar-te, a crear coses noves i a trobar noves solucions.

Aquí comença el teu camí. Som-hi!

1.2 INSTAL·LACIÓ I FUNCIONAMENT DE L'INTÈRPRET

Python és un llenguatge que necessita un intèrpret per tal que l'ordinador entengui i executi les instruccions del codi. Així que, el primer que haurem de fer és instal·lar-lo. A continuació, s'expliquen els passos per fer-ho:

Per a ordinadors Mac:

És possible que tinguis instal·lat, per defecte, un intèrpret de Python. Pots comprovar-ho anant a la terminal (recorda que pots accedir-hi més ràpidament fent ús de la drecera Cmd + espai + terminal). Allà, a la terminal, només cal que escriguis *python*. Tant si no el tens instal·lat (en aquest cas, t'apareixerà un missatge dient que no es troba), com si tens una de les versions 2.7, és aconsellable instal·lar el nou intèrpret que es recomana tot seguit, doncs la versió de Python 2 va deixar de tenir manteniment al gener del 2020.

Ves a https://www.python.org/downloads/mac-osx i fes clic a la versió més recent de Python 3, la qual apareix just a sota de Python Releases for Mac OS X (important: assegura't de no fer clic a la versió de Python 2). Quan ets a la pàgina nova, ves a la taula de sota i selecciona el fitxer que s'adapti a la versió del teu sistema operatiu, que serà un d'aquests dos:

macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	5a95572715e0d600de28d6232c656954	34479513	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	4ca0e30f48be690bfe80111daee9509a	27839889	SIG

Important: si es disposa de la versió del sistema operatiu 10.9 (Mavericks) o superior, es recomana descarregar l'instal·lador macOs 64-bit. Guarda el fitxer, ves a la secció de descàrregues del navegador i fes-hi clic. Segueix els passos que et marca la finestra que apareix.

Per a ordinadors Windows:

Si treballes amb Windows també pots comprovar si tens un intèrpret Python instal·lat. Per veure-ho, obra Command Prompt i escriu *python.* De la mateixa manera que per a usuaris de Mac, si no el tens instal·lat o tens una versió 2.7, és millor instal·lar l'intèrpret 3.7.

En aquest cas, has d'anar a https://www.python.org/downloads/windows i fer clic a la versió més recent de Python 3, tal com s'indica a l'apartat de Mac. Els passos són els mateixos fins aquí. Després, i una vegada siguis a la taula de sota, pots descarregar-te un d'aquests quatre fitxers per iniciar la instal·lació:

Windows x86 executable installer	Windows		ebf1644cdc1eeeebacc92afa949cfc01	25424128	SIG
Windows x86 web-based installer	Windows		d3944e218a45d982f0abcd93b151273a	1324632	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	a2b79563476e9aa47f11899a53349383	26190920	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	047d19d2569c963b8253a9b2e52395ef	1362888	SIG

Per a Windows hi ha més opcions, ja que es fa diferència entre aquells ordinadors que implementen l'estructura Intel 64, o més formalment coneguda com x86-64, i els que no. A més a més, ara també hi ha l'opció de poder descarregar un instal·lador *web-based*, que descarregarà els components que calguin durant la instal·lació.

Una vegada el tinguis instal·lat, torna a obrir la terminal (en Mac) o Command Prompt (en Windows) i torna a escriure *python.* Si Python 2 ja està instal·lat al teu ordinador, la manera d'executar la nova versió de Python 3 és amb l'ordre python3 en lloc de python.

Comprova que, efectivament, la nova versió instal·lada correspon a la més recent i escriu *import this*. Hauràs de veure en pantalla com apareixen els lemes que formen *The Zen of Python.* També pots fer una petita operació. Escriu, per exemple *5 * 3.* T'apareix 15? Doncs, ja has fet servir Python com a calculadora, així de senzill i ràpid.

En el cas que tinguis problemes per instal·lar l'intèrpret o no et funcioni correctament i no vulguis perdre massa temps per mirar de trobar on hi ha l'error, pots fer ús de Python sense necessitat d'instal·lar res. Per fer-ho, adreça't a aquesta pàgina web: http://www.repl.it/. Aquí, podràs treballar de manera totalment gratuïta i seguir totes les lliçons del curs; només t'has de registrar amb una adreça de correu. Aquesta plataforma, a més, et permetrà accedir als teus exercicis des de qualsevol lloc o ordinador, a través del núvol (*Cloud Computing*).

Després, hauràs de seleccionar Python com el teu llenguatge de programació. Si ja fas servir un altre llenguatge de programació com Java, Swift o d'altres, també pots fer servir aquesta plataforma per escriure codi.

1.3 INSTAL·LACIÓ DE L'INTÈRPRET

Hola a tothom!

Segur que ja teniu moltes ganes de començar a programar amb Python. Abans d'això, però, necessitem instal·lar l'intèrpret. Aquest vídeo mostra els passos de la instal·lació per a un ordinador Mac, tot i que els usuaris i usuàries de Windows també el podran seguir, ja que els passos a seguir són pràcticament els mateixos.

Primer de tot, ens adrecem a https://www.python.org/ i fem clic a *downloads*. Aquí veiem que es mostren les opcions per instal·lar l'intèrpret amb Mac o amb Windows i, també amb Linux/Unix. En aquest cas, fem clic a *Latest Python 3 release* i ens dirigim directament a la taula de sota de la pàgina nova, on trobem, no només les opcions per descarregar per a Mac, sinó també per a Windows.

Farem clic en aquest fitxer (macOS 64-bit installer), ja que la versió del sistema operatiu d'aquest ordinador s'adapta als requisits corresponents. Recorda només que, si tens Windows, has de seleccionar el fitxer que et convingui més. Per a Mac amb una versió inferior al 10.9, caldrà instal·lar el de 32-bit.

Fem clic en el fitxer i ens apareix la finestra per començar a descarregar l'instal·lador de l'intèrpret. Guardem el fitxer, anem a l'apartat de descàrregues, el seleccionem i s'obre la finestra de benvinguda. Cliquem a "Continuar", ens apareix la secció *Read Me*, cliquem de nou a "Continuar" i així arribem a l'apartat de la llicència. Fem clic a "Continuar" una vegada més. Acceptem la llicència d'ús. Després, seleccionem a quin disc volem guardar l'intèrpret.

Finalment, hem d'escollir on volem guardar el fitxer dins del disc, tot i que el sistema ja ofereix una opció per defecte per fer-ho. Fem clic a instal·lar i esperem uns instants a què es completi l'acció. Veiem que la barra està en progrés. Sí, ara ja està tot llest i podem tancar la finestra de l'instal·lador.

Ara només cal que anem a la terminal, per comprovar que l'intèrpret de Python està instal·lat correctament**.** Si tot ha anat bé, el sistema ho ha de reconèixer amb un missatge, que indica la versió de Python instal·lada.

Ja ho tenim. Fins aviat!

1.4 INSTAL·LACIÓ I FUNCIONAMENT DE L'IDE (L'ENTORN DE PROGRAMACIÓ)

Un entorn de programació (també anomenat IDE) és un editor de codi que permet fer ús de totes les funcionalitats que ofereix un llenguatge de programació d'una manera més àgil, visual, eficaç i ordenada.

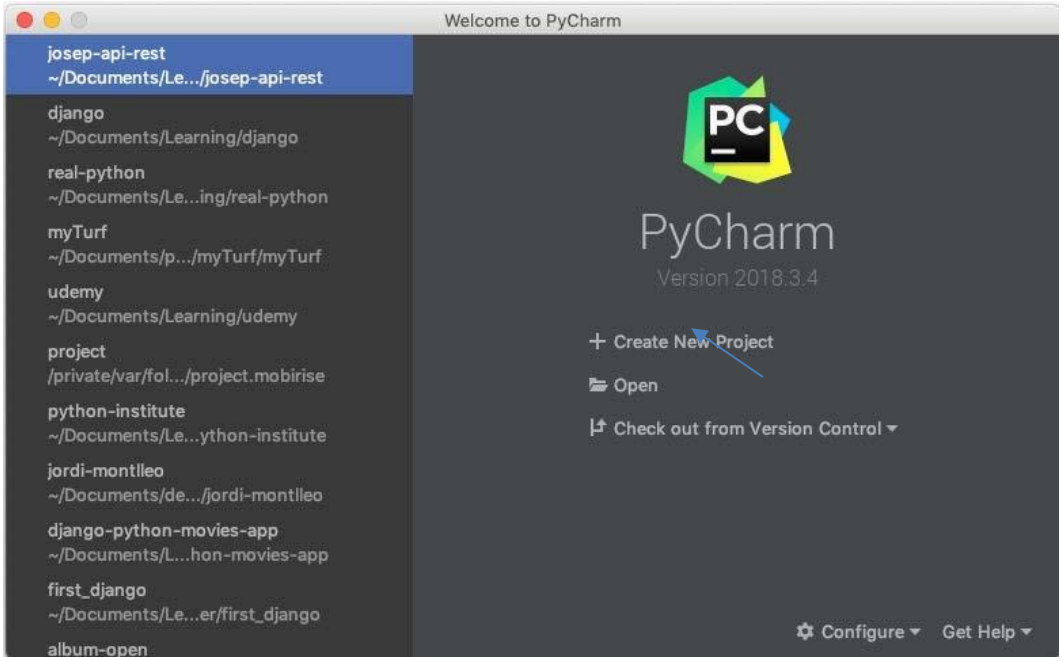
Hi ha editors de codi que poden ser més específics per Python, com ara PyCharm, Spyder o Thonny i, d'altres que no ho són, però que també permeten fer ús de Python, com per exemple Eclipse, Sublime Text, Atom, GNU Emacs, Vi/Vim o Visual Studio.

Durant els exercicis del curs, farem servir PyCharm per editar el codi i fer córrer el fitxer. En qualsevol cas, si ja tens experiència amb altres editors que suportin Python, pots fer-los servir en comptes de PyCharm. Qui treballa en programació, quan s'enamora d'un editor és difícil que el canviï per un altre. Són moltes hores treballant i es crea un vincle. Els programadors i programadores som així!

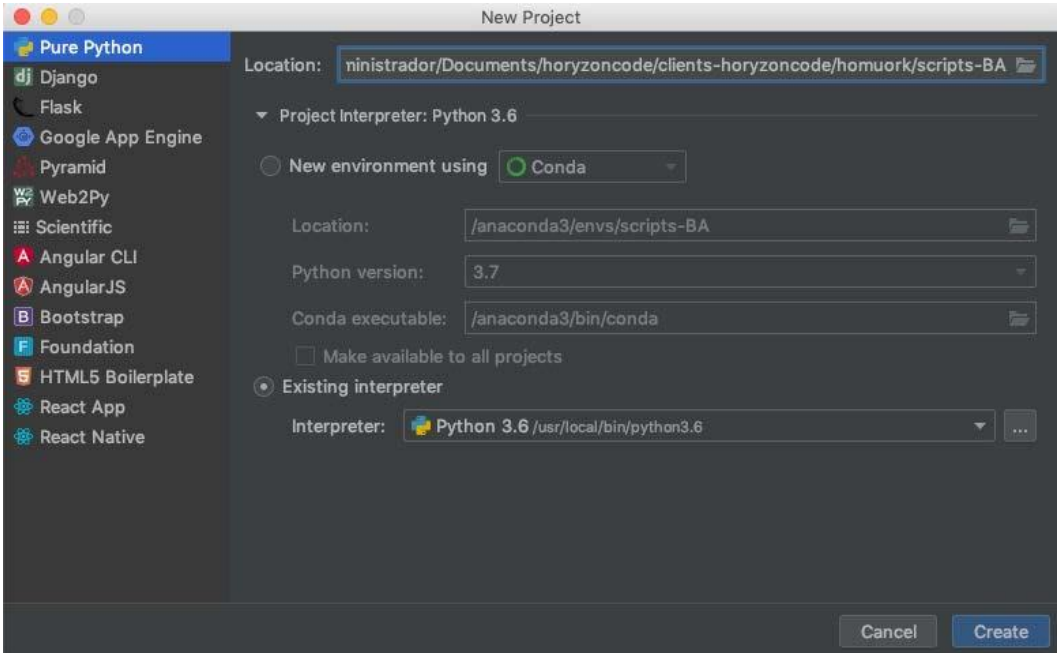
Passem, doncs, a instal·lar la versió gratuïta de PyCharm. Primer, només cal que ens dirigim directament cap al web del proveïdor: https://www.jetbrains.com/pycharm/. Allà seleccionem quina versió volem descarregar, depenent del sistema operatiu i de si volem la versió Professional o la Community. Aquesta segona és la gratuïta. A continuació, ens pregunten per l'adreça de correu per si volem rebre contingut educatiu i també t'hauria d'aparèixer una finestra per descarregar el fitxer de l'instal·lador del programa. Una vegada completada la descàrrega, seguirem els passos per instal·lar finalment el programa.

Important: recordem que, si volem estalviar-nos la instal·lació (o tenim problemes de compatibilitat), podem programar amb Python sense necessitat de tenir l'intèrpret ni l'IDE. Ho podem fer directament des de la xarxa, a través de http://www.repl.it, una aplicació web gratuïta.

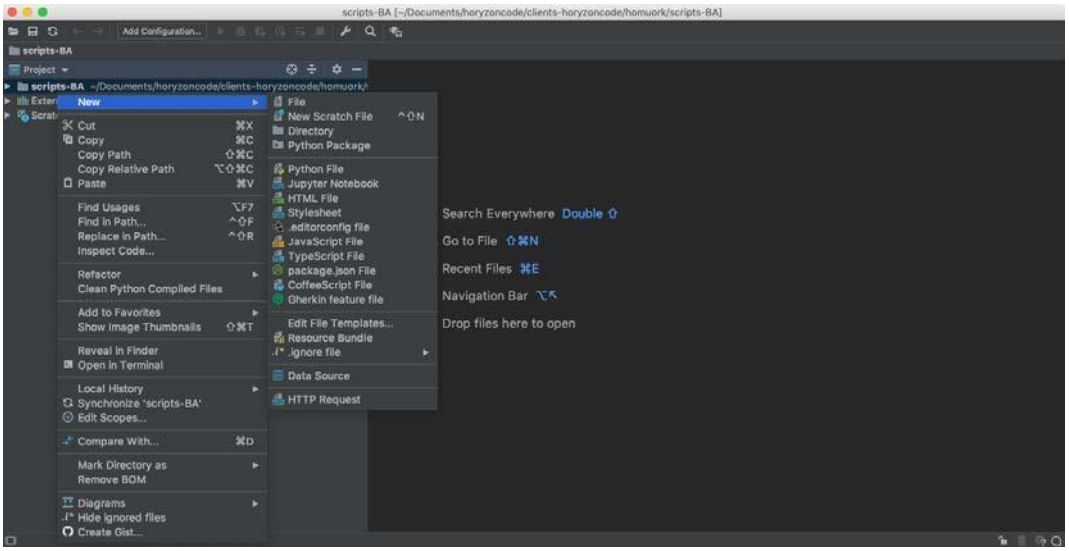
El següent pas serà obrir PyCharm i crear un projecte nou en el qual hi guardarem tots els exercicis que anirem fent durant el curs.



Tot seguit, seleccionem on volem desar el projecte des de *Location*, marquem l'opció d'*Existing Interpreter*, on hauria d'aparèixer la versió de l'intèrpret instal·lat, i fem clic a *Create.*



Amb el projecte obert, anirem al menú vertical de l'esquerra. Fent clic amb el botó dret, s'obrirà la carpeta del projecte. Així podrem crear un fitxer nou des d'on escriurem el nostre codi (*New -> Python File*).

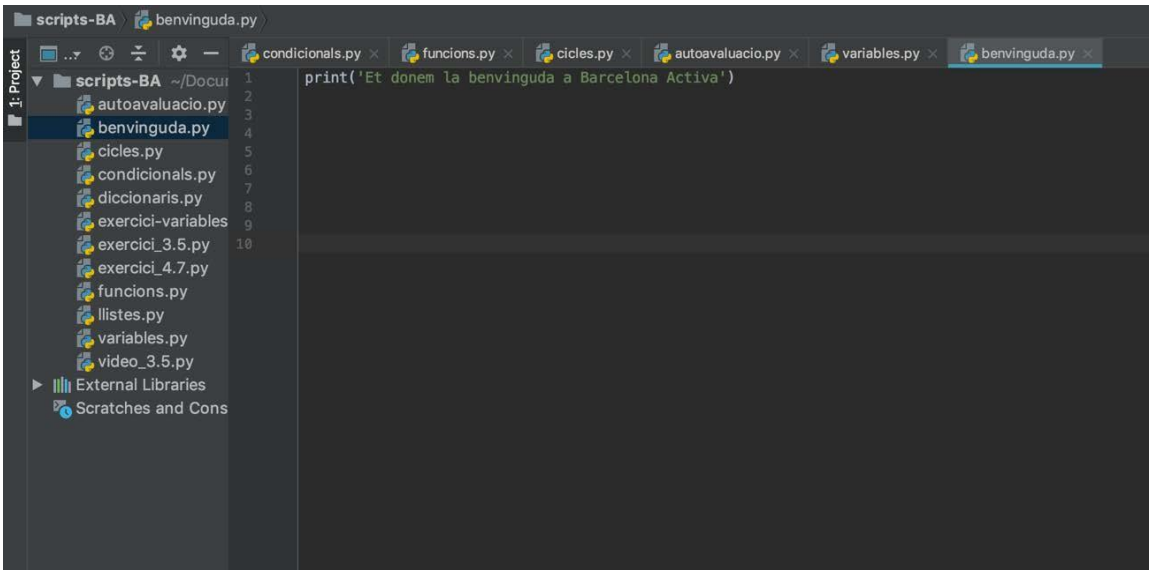


PyCharm té moltes funcions i potser espanta una mica la primera vegada, però, per començar a programar, només cal aprendre un parell d'eines. El programa el farem servir,

principalment, com a editor pur, per escriure el codi i fer les entrades i imprimir els resultats a través de la consola. D'aquesta manera, només ens haurem de centrar en aprendre el llenguatge Python.

Per comprovar que tot funciona correctament, escriurem el codi següent dins de la caixa de text, tal com apareix a la imatge de sota. Per tal d'obtenir un resultat, només caldrà fer clic a *play* (per executar l'operació *Run*) allà on marca el cercle.

print('Et donem la benvinguda a Barcelona Activa')



El que fa l'operació *Run* és convertir el codi escrit a la caixa de text de l'editor en llenguatge màquina per tal que l'ordinador entengui les instruccions. A la part de sota hi ha una altra caixa de text que mostra els resultats a través de la consola, una vegada l'ordinador ha executat les ordres que se li han donat. Cada vegada que canviem el codi dins de la caixa de l'editor i fem clic a *Run,* la consola mostra un resultat nou, seguint les instruccions noves.

Fins ara, hem fet servir una operació molt important anomenada *print*, pròpia del llenguatge Python, la qual és l'encarregada de fer sortir per la consola (o pantalla) el contingut entre els parèntesis.

Tot just acabem de començar. El teu primer algoritme està cada vegada més a prop.

1.5 INSTAL·LACIÓ DE L'IDE

Hola!

Per poder aprofitar totes les funcionalitats del llenguatge Python d'una manera més àgil i completa, necessitarem un entorn de programació que ens faciliti aquesta tasca.

Trobem diverses opcions. Unes són més específiques de Python i d'altres són més generalistes. Per a aquest curs, escollirem PyCharm com el nostre entorn de programació. Si ja treballes amb un altre IDE (recordeu que IDE fa referència al concepte d'entorn de programació) pots utilitzar-lo, sempre i quan aquest IDE suporti el llenguatge Python. No cal que canviïs la teva manera de treballar.

Dit això, ens adrecem al web des d'on podrem descarregar PyCharm (https://www.jetbrains.com/pycharm/). La instal·lació és prou senzilla. Escollim si volem la versió de Windows, Mac o Linux, i ens apareix una finestra emergent just aquí. Guardem el fitxer i esperem uns instants a què la descàrrega finalitzi. Una vegada tinguis descarregat el fitxer de l'instal·lador, l'obres i, si ets usuari de Mac, només cal que l'arrosseguis fins a la carpeta d'aplicacions.

Ara toca obrir PyCharm. Per començar, creem un projecte nou. Podem triar el nom més significatiu per a nosaltres, com per exemple la temàtica que abordarà, i seleccionar-ne la ubicació. Un cop creat ens demana quin intèrpret volem fer servir. Seleccionem Existing Interpret i ja tenim el nostre entorn llest per començar a treballar.

No pateixis per totes les eines i funcions que presenta PyCharm. Només hauràs d'aprendre'n un parell. De moment, ves aquí dalt a l'esquerra, on veuràs una carpeta amb el nom que has donat al teu projecte. Fes clic al damunt seu amb el botó dret i selecciona "New Python File". Ja tenim un fitxer llest per començar a picar codi.

Comencem! En aquesta caixa de text. i escrivim aquesta línia de codi: print('Et donem la benvinguda a Barcelona Activa'). Per comprovar que tot funciona correctament, només queda executar-la, tot fent clic al play.. I, sí, aquí a la consola obtenim el resultat que volíem, que era fer sortir per pantalla el text que hi ha dins del parèntesis print.

Com ho veus? Ja has executat el teu primer programa de Python i t'has comunicat amb l'ordinador. I només és l'inici. Seguim!

1.6 IDEES CLAU: QUÈ ÉS PYTHON I COM COMENÇAR A PROGRAMAR

Python és un dels llenguatges de programació que està guanyant més popularitat arreu del món. Ofereix un gran ventall d'aplicacions, entre les que destaquen:

- Desenvolupament web
- Creació de vídeojocs
- Anàlisi de dades i *big data*
- *Machine learning* i intel·ligència artificial
- Automatització de tasques

Les característiques principals que descriuen Python són:

- És un llenguatge de codi lliure.
- Està basat en l'orientació a objectes.

- És fàcil d'aprendre.
- És un llenguatge interpretat i multiplataforma.
- S'anima als programadors i programadores a dissenyar el seu codi seguint els lemes de *The Zen of Python.*

L'intèrpret és l'encarregat de traduir el teu codi a llenguatge màquina per tal que l'ordinador executi les teves instruccions. Aquest intèrpret és de fàcil instal·lació i només cal seguir les instruccions per a la seva descàrrega, des del web https://www.python.org/downloads/.

Per tal de poder aprofitar totes la funcionalitats de Python d'una manera més àgil i ordenada, és convenient fer ús d'un entorn de programació, també anomenat IDE. En el nostre cas, s'ha escollit PyCharm. Per poder fer la descàrrega, només cal anar al web

https://www.jetbrains.com/pycharm/ i iniciar els passos fins a finalitzar el procés.

En el cas que es volgués començar a programar sense necessitat de descarregar l'intèrpret ni l'IDE, es pot escriure el codi i executar-ho des d'internet a través de l'aplicació gratuïta https://www.repl.it.

Per tal de començar a executar codi, a la finestra de l'editor de text escriurem el nostre primer codi: *print('Aquí hi aniria una frase')* i el farem córrer per tal que l'intèrpret tradueixi les nostres instruccions a llenguatge màquina. El resultat s'imprimirà per pantalla en la finestra de la consola.

2 LES VARIABLES

Hola a tothom!

Ha arribat el moment de fer ús de les variables amb Python. Primer, veurem com escriure-les correctament i de què es componen, quin tipus d'informació emmagatzemen i quines accions podem fer servir per operar amb les seves dades o contingut.

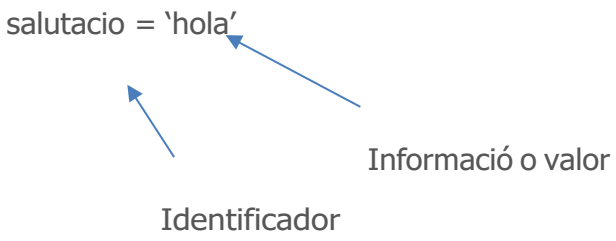
Les variables són les entitats que permeten emmagatzemar la informació dels programes. Són, doncs, principalment, espai de memòria on podem accedir per fer ús de les dades que hi guardem. En aquest curs, però, no ens aturarem a analitzar la gestió de la memòria, sinó que ens centrarem a saber treballar amb les variables des d'un punt de vista pràctic i molt orientat a la redacció de codi.

La sintaxi de les variables amb Python és molt senzilla i neta. Començarem per gestionar els tipus de variables més bàsics i, després, veurem algunes variables menys intuïtives que, potser, requeriran més atenció si mai abans no hem après cap llenguatge de programació.

2.1 LES DIFERENTS VARIABLES I LES SEVES OPERACIONS

En aquest article explorarem què són les variables, descriurem les seves característiques i explicarem les operacions que podem realitzar-hi. Bàsicament, les variables serveixen per assignar un espai de memòria a l'ordinador per tal de preservar una informació o, millor dit, per preservar el contingut d'un objecte. A Python, qualsevol entitat és un objecte.

Així, doncs, una variable respecte al codi estarà formada pel seu identificador (nom assignat) i el valor o dada de la informació (objecte) que contingui. Exemple:



A continuació, exposarem els diferents tipus d'objectes més habituals que pot guardar una variable. Són:

Cadena de caràcters (*strings*)

Aquí trobaríem les paraules, els textos, adreces de correu o pàgines web (URL) o, qualsevol altra informació que contingui una cadena de caràcters. De fet, qualsevol cadena de caràcters que faci referència a una informació i que estigui continguda entre cometes simples o dobles serà entesa per Python com una *string.*

Exemples:

```
Ciutat naixement = 'Barcelona'
llengua = 'Català'
clau_acces = "htrn!5647@#GF[[}}**"
url = 'http://www.barcelonaactiva.cat'
```

- Números enters (*int*)

  Són aquells números positius i negatius que no tenen decimals. S'inclou també el zero.

  Exemple:

```
edat = 21
errades = 0
```

- Números decimals (*float*)

Són aquells que tenen decimals, ja siguin positius o negatius.

Exemples:

```
puntuacio_mitjana = 3.50
```

- Veritat o fals (*boolean*)

Aquest tipus d'informació només pot tenir el valor de veritat (*True*) o fals (*False*).

Exemple:

```
inscripcio_realitzada = True
pagament_efectuat = False
```

- Altra informació, com poden ser les funcions.

Exemple:

```
primera_accio = obrir
segona_accio = tancar
```

Una nova variable pot contenir el valor de la informació d'una altra. Així que, per exemple, podem escriure el següent:

```
municipi_actual = ciutat_neixament
```

El valor que conté la variable *municipi_actual* és 'Barcelona' (una cadena de caràcters o *string*) i el *pes_minim* té un valor assignat de 3,50 (un número amb decimals o *float*).

Nota:

A partir d'ara, farem servir el nostre editor de codi i veurem resultats per pantalla. Només cal saber el següent:

print() – mostra per pantalla el valor que conté dins dels parèntesis.

També és important que sàpigues que, si vols afegir comentaris al teu codi per tal de fer aclariments al respecte, ho pots fer. Només cal començar la línia amb un símbol # o escriure el text entre "'''...'''".Aquests comentaris no seran executats per l'intèrpret i ens serviran perquè els nostres companys i companyes que treballen amb nosaltres en el mateix projecte puguin seguir més fàcilment el nostre desenvolupament. Els programadors i les

programadores treballen normalment en equip. És així com es poden arribar a crear grans projectes.

Una vegada hem definit el tipus d'informació o objectes amb els quals treballarem i els hi hem assignat un espai a la memòria mitjançant una variable, vegem quin tipus d'operacions permeten fer.

- Enters (*int*) i números decimals (*float*)
  Els números permeten fer qualsevol de les operacions matemàtiques dels seus grups. Les més bàsiques són la suma (+), la resta (-), la multiplicació (*), la divisió (/), etc.

  Exemples:

```
"""calculem la nota mitjana dels alumnes"""
joan = 8.3
carla = 9.2
adria = 7.5
dolors = 6.9
suma_notes = joan + carla + adria + dolors
n_alumnes = 4
mitjana = suma_notes/n_alumnos
print(mitjana)
```

Important:

Les operacions segueixen l'ordre de jerarquia convencional.

https://ca.wikibooks.org/wiki/Matem%C3%A0tiques_(nivell_ESO)/Jerarquia_de_les_operacions

A Pyton, els decimals s'escriuen sempre amb un punt i no amb coma.

Sobre les divisions, cal destacar que, si dividim dos números enters, el resultat sempre serà reconegut per Python com un número del tipus *float*, tot i que la divisió doni un resultat enter. Exemple:

```
alumnat = 10
grups= 2
alumnat_per_grup = alumnat / grupos
print(alumnat_per_grup)
print(alumnat_per_grup)
```

El resultat, tot i ser un 5 rodó, Python el reconeix com a 5.0.

- Cadena de caràcters:

Les cadenes de caràcters també permeten la suma i la multiplicació. Comprovem-ho amb un exemple:

```
# sumem dues paraules i  després les multipliquem per 2
p1 = 'hola'
p2 = 'barcelona'
salutacio = p1 + ' ' +
p2print(salutacio)
print((salutacio,) *
```

Veuràs que un espai també pot ser representat com a caràcter. A la funció *print(),* es pot imprimir una seqüència d'*strings* separada per comes. Aquestes comes representaran un espai a la pantalla.

Com acabes de veure, la funció *print()* accepta operacions dins dels seus parèntesis. Això vol dir que, primer, resol l'operació i, després, n'imprimeix el resultat.

Amb això ja pots començar a escriure codi. Queda't amb nosaltres i aprendràs encara molt més.

2.2 PARTICULARITATS DE LES VARIABLES

Hola a tothom!

Seguim amb les variables i les seves operacions. Ara veurem com entrar per teclat una informació, per tal de guardar-la en una variable.

Escrivim aquest petit codi:

```
salutacio = 'Hola'
print('el teu nom:')
nom = input()
print(salutacio, nom)
```

Sí, fer entrades per teclat és així de senzill. Només hem de tenir en compte que les entrades des d'*input()* sempre són gestionades com a cadenes de caràcters (*strings*). És com si nosaltres sempre entréssim la informació entre cometes. Per exemple:

```
n1 = 5
print('altre numero:')
n2 = input()
print(n1 + n2)
```

Per teclat, introduïm 5 i... apa!, ja tenim un error. Però és normal, ja que hem intentat sumar un número enter (int) amb una *string* (str), i això no es pot fer. Recorda que totes les entrades (input()) seran *strings*. T'estaràs preguntant com podem operar amb números, oi? Doncs bé, tenim mecanismes per modificar el tipus d'informació amb la qual estem treballant. Mira, si n2 el posem dins dels parèntesis d'int(), Python convertirà el valor introduït, que és de tipus *string*, en un número enter. I ara sí que podem fer la suma.

Bé, no? Cada vegada podem fer més coses amb Python.

Per tal que un número enter es converteixi en una *string* ho fem així. Hem posat el valor dins dels parèntesis d'str(), d'aquesta manera:

```
x = 8
y = str(x)
```

I, si ho necessitem, podem canviar un número *float* per un número enter.

```
x = 10
y = 5
z  =  10/5
resultat = int(z)
print(resultat)
```

El resultat que obtenim no és 2.0 (un *float*), sinó que obtenim 2 (un *int*). Òbviament, també podem passar de *float* a *string* i viceversa. Recorda que les divisions donen sempre valors del tipus *float*.

Ara revisem els *boolean* (valors que només poden ser veritat o fals). Ells també es poden convertir en *int* o *float*. Ho fem així:

```
resposta1 = True
resposta2 = False
print(int(resposta1))
print(float(resposta2))
```

Sí, efectivament. El valor *True* quan passa a número enter és 1 i el valor *False* passa a 0. Si es passa a *float*, doncs True és 1.0 i False és 0.0.

Finalment, si vols saber exactament quin tipus d'informació conté una variable, només cal que escriguis aquesta sentència:

```
x = 10.5
y = 'pilota'
print(type(x))
print(type(y))
```

Efectivament, *x* conté un valor del tipus *float* i *y* en conté un de tipus *string*.

Fins el proper vídeo!

2.3 ARA ET TOCA A TU: REPTE. CALCULA EL VALOR DE X

Suposem que fem diverses compres a l'any, en total quatre, a una empresa proveïdora que ens ven els seus productes en dòlars. Els imports de les compres són: 356.75 $, 487.45 $, 295.83 $ i 532.00 $. Haurem de convertir el valor de dòlars a euros.

Part 1: Calcula i imprimeix per pantalla les dades següents:

- Suma total de les compres en euros
- Mitjana de les quatre compres en euros

Nota: procura que la variable que necessitaràs per tenir el canvi de dòlars a euros rebi la dada a través d'una entrada per teclat.

Part 2:Imagina que l'última comanda (la de 532.00 $) ha tingut un problema i només arriba el 80 % del gènere en bones condicions i, per tant, diem al proveïdor que només li pagarem el 80 % del'import acordat prèviament.

- Calcula, ara, els mateixos resultats que en el cas anterior i imprimeix-los per pantalla.
- Podries dir quin tipus de dada (informació) és la suma total de qualsevol dels dos casos?
- Sabries canviar-la a una dada del tipus *string*?

Procura esforçar-te per resoldre les dues parts de l'exercici sense necessitat de mirar la solució. Si no te'n surts, llavors consulta-la.

Endavant!

2.4 SOLUCIÓ EXERCICI

Abans de començar a picar codi, és important entendre què ens demana el problema i saber mentalment quins passos haurem de seguir per tal d'arribar a la solució òptima. En programació, és important, no només fer que el teu codi funcioni, sinó que també resolgui el problema de la manera més eficient possible. Recorda que pots repassar els lemes de *The Zen of Python* per orientar-te.

En aquest cas, el primer de tot és crear les variables i guardar allà els imports de les quatre compres i el canvi de dòlars a euros. Assumirem que el canvi de dòlars a euros és de 0,890.

```
c1 = 356.75
c2 = 487.45
c3 = 295.83
c4 = 532.00
print('Introdueix canvi:')
canvi = input() # canvi dollars a euros
canvi = float(canvi)
```

Nota: recorda que tots els *inputs* entren com a *string* i els hem de canviar a *float* per fer operacions amb les compres.

Tot seguit, podem calcular la suma de les compres i imprimir-ne el resultat:

```
suma_euros = (c1 + c2 + c3 + c4) * canvi
mitjana = suma_euros / 4
```

```
print('Suma de les compres {:.2f} €'.format(suma_euros))
print('Mitjana de les compres: {:.2f} €'.format(mitjana))
```

Resultat de la suma = 1.488, 11 €

Resultat de la mitjana = 372,03 €

Ara, resoldrem el segon cas en el qual només hauràs de pagar el 80 % de l'import de la quarta compra:

```
suma_euros_2 = (c1 + c2 + c3 + c4*0.8) * canvi
mitjana_2 = suma_euros_2 / 4
```

```
print('Nova suma: {:.2f}'.format(suma_euros_2))
print('Mitjana de les compres: {:.2f} €'.format(mitjana_2))
```

Nota: fixa't en el codi del quadre de dalt. Dins dels { } s'introduirà el valor de la variable que està en format() i s'escriurà amb dos decimals. Això servirà per tal de reduir els decimals a només dos.

Resultat de la suma nova = 1.393,41 €

Resultat de la mitjana nova = 348,35 €

Finalment, per saber quin tipus de dada és la suma total (qualsevol dels dos casos) i passar-la a una de tipus *string,* hem de fer-ho així:

```
suma_type = type(suma_euros)
suma_string = str(suma_euros)
```

2.5 IDEES CLAU: LES VARIABLES

A continuació, farem un repàs de les idees més importants que hem tractat.

- Què és realment una variable.
- Com declarar-la en el codi.
- Quins tipus d'informació poden guardar les variables.
- Quines operacions podem fer amb els diversos tipus de dades.
- Com podem passar un tipus de dada a un altre.

També hem après a entrar una dada per teclat i guardar-la en una variable. A continuació, hem estat capaços d'imprimir resultats en pantalla i, fins i tot, especificar quants decimals volem mostrar.

I, no menys important, ara sabem escriure comentaris en el nostre codi. Aquests comentaris no seran executats per l'intèrpret i ens serviran perquè la gent del nostre equip pugui seguir més fàcilment el nostre desenvolupament.

3 ESTRUCTURA DE DADES. LLISTES I DICCIONARIS

Ha arribat el moment de treballar amb aquests dos tipus d'estructures de dades: les llistes i els diccionaris. També aprofundirem sobre les accions que podem fer amb elles, com ara afegir o treure elements, accedir a la informació, etc.

Les llistes són les entitats (objectes) que permeten guardar una col·lecció de dades dins d'una mateixa variable. Els diccionaris també ho són, però en aquest cas necessitem una clau per accedir a la informació.

Durant la redacció de codi és molt habitual fer ús d'aquestes estructures, així que aquest mòdul és prou important per tal que, una vegada haguem entès com treballar-hi, fem un salt significatiu com a futurs programadors i programadores.

3.1 TREBALLAR AMB LLISTES

Les llistes són una col·lecció de dades (objectes) que poden ser del mateix tipus o no. Aquestes llistes, doncs, permeten emmagatzemar qualsevol tipus de dades de Python: *int*, *float*, *string*, *boolean* i qualsevol altre objecte. Recorda, a Python tot són objectes.

Les llistes són emmagatzemades a la memòria, quan són assignades a una variable. A continuació, s'exposa una llista amb una col·lecció de números enters *(int)*, la qual és assignada a una variable anomenada *llista_enters*:

```
llista_enters = [4,6,3,7,8]
```

Les llistes exposen les seves dades separades per comes i la col·lecció sencera ha d'estar dins de dos claudàtors: un a l'inici i l'altre al final. Aquí es mostra un altre exemple. En aquest cas, la mateixa llista inclou dades de diferents tipus.

```
llista_mixta = [-4, 7.8, 'hola', False]
```

Cada dada de dins d'una llista té assignada una posició. La primera posició és la que està més a l'esquerra. El seu índex és 0. La següent posició és 1, després 2 i així successivament fins a l'última dada que es troba més a la dreta. Per poder accedir a una dada dins de la col·lecció i guardar-la en una variable, ho fem així:

Important: la primera posició és sempre 0 i no pas 1.

```
a = llista_enters[0]
b = llista_mixta [3]
```

Fixa't que el valor de *a* és 4 i el valor de *b* és *False*. Si vols imprimir les dades d'una col·lecció sense necessitat de crear una variable, ho pots fer directament de la manera següent:

```
print(llista_mixta[2])
```

El resultat que mosta la pantalla és *hola*.

Aquesta selecció també es pot fer indicant l'índex corresponent començant des de la dreta. En aquest cas, s'utilitzen números negatius. L'última posició té assignat l'índex -1. A partir d'aquí, cap a l'esquerra, els índexs són -2, -3, etc.

Per exemple, si volem extreure de *llista_mixta* la dada *7.8*, ho podem fer d'aquestes dues maneres:

```
a = llista_mixta[1]
b = llista_mixta[-3]
```

Si vols saber la quantitat de dades que conté una llista, ho pots fer cridant la funció *len(),* tal com es veu en l'exemple:

```
print(len(llista_enters))
```

Aquest codi permet imprimir per pantalla la quantitat d'objectes que conté *llista_enters*, que té fins a cinc valors.

Tot seguit, exposem les accions amb les que podem gestionar les dades d'una llista.

- Afegir una nova dada – *append()*:

```
llista_mixta.append(True)
```

La sintaxi d'aquesta acció és compon de la manera següent: primer, s'escriu el nom de la variable que fa referència a la llista en qüestió, tot seguit s'hi posa un punt i, just a la dreta, s'escriu l'acció d'afegir. *Append* indica l'acció d'afegir la dada que està dins dels parèntesis.

Important: la dada o objecte afegit a una llista mitjançant *append()* sempre es col·locarà en l'última posició (la que està més a la dreta).

- Extreure l'última dada – *pop()*:

Per eliminar l'última dada afegida, cridarem la funció *pop(),* seguint la mateixa sintaxi que amb l'acció *append(),* tot i que, per a aquesta nova funció, no cal afegir res en els parèntesis. Posem un exemple:

```
llista_enters.pop()
print(llista_enters)
```

Si ara imprimeixes la *llista_enters* per veure quines dades mostra, veuràs que l'última posició ha estat eliminada.

- Identificar la posició d'una dada – *index():*

Si vols saber la posició que ocupa una dada dins de la col·lecció, hauràs d'executar aquesta acció, seguint el mateix model que les altres dues anteriors:

```
print(llista_mixta.index('hola'))
```

Com podem veure, dins dels parèntesis, hi ha d'anar la dada de la qual vols saber la posició que ocupa a la llista.

- Fer una còpia d'una llista, seleccionant només certes posicions:

Si vols fer servir només part d'una col·lecció i guardar-la en una altra variable, hauràs de seguir el mètode següent:

```
a = llista_enters[0:3]
```

La variable a conté una llista que és una còpia de *llista_enters* des de la posició 0 fins a la posició 2. L'índex de la dreta no s'hi inclou. Podem posar un altre exemple:

```
notes_alumnes = [8.75, 3.50, 5.50, 6.25, 9.75, 2.75, 4.50, 7.60]
notes_ultimes = notes_alumnes[3:]
```

notes_ultimes tindria els valors: 6.25, 9.75, 2.75, 4.50 i 7.60.

Si l'índex de la dreta el deixem en blanc, l'intèrpret entendrà que ha d'agafar totes les dades des de la posició de l'esquerra fins al final. Si, en canvi, deixem en blanc l'índex de l'esquerra, la còpia començarà des de la posició 0. Veiem aquest últim exemple:

```
notes_ultimes = notes_alumnes[:5]
```

Ara *notes_ultimes* tindria aquests valors: 8.75, 3.50, 5.50, 6.25 i 9.75.

Recorda que l'índex de la dreta no hi està inclòs i que, per tant, l'última posició és la que es troba una abans.

3.2 TREBALLAR AMB DICCIONARIS

Els diccionaris són una col·lecció de dades que permeten accedir a la informació que contenen mitjançant una clau. Aquest tipus d'estructura de dades també es coneix per clau-valor. A continuació, es mostra un exemple per veure l'estructura i com es pot escriure un diccionari.

```
notes_dict = {'Joan': 4.85, 'Miquel': 7.80, 'Meritxell': 8.15, 'Laura': 4.75}
```

També podem crear diccionaris amb el mètode dict. Mostrem un ejemple:

```
notes_dict = dict(Joan= 4.85, Miquel= 7.80, Meritxell= 8.15, Laura= 4.75)
```

Aquí tenim, doncs, el nom de la variable *notes_dict* on es guarda el diccionari. Les claus són 'Joan', 'Miquel', 'Meritxell' i 'Laura'. Els valors són 4.85, 7.80, 8.15 i 4.75. També veiem que, per obrir i tancar un diccionari, es fan servir els caràcters { }. Les llistes, en canvi, recorda que fan servir els claudàtors [].

Si volem accedir, per exemple, a la nota de la Meritxell i guardar-la en una variable, ho farem així:

```
nota_meritxell = notes_dict['Meritxell']
```

Important: els diccionaris accepten qualsevol tipus de dada com a valors. Per a les claus, normalment es fan servir *strings* o números enters.

També podem accedir al valor d'una clau determinada fent ús de la funció *get()*. Es faria així:

```
notes_dict.get('Laura')
```

Aquesta línia de codi ens permet accedir al valor 4.75.

Si volem afegir una nova parella clau-valor, només haurem d'escriure aquesta sentència:

```
notes_dict['Eduard'] = 6.80
```

Ara, el diccionari té una nova clau anomenada 'Eduard' amb un valor de 6.80.

Per saber el número de clau-valor que conté un diccionari, ho podem fer de la mateixa manera que amb les llistes: hem de cridar la funció *len()* i posar a dins dels parèntesis el nom de la variable que conté el diccionari:

```
print(len(notes_dict))
```

En aquest cas, ens apareixerà per pantalla el número de registres clau-valor perquè hem cridat la funció *len()* dins de la funció *print().*

Si volem, per exemple, extreure la col·lecció de claus en forma de llista, ho podem fer amb dos passos:

```
claus = notes_dict.keys()
claus_llista = list(claus)
```

La funció *keys()* està associada als diccionaris. Aquesta funció genera una col·lecció que conté les claus del diccionari, la qual es pot convertir en una llista fent ús de la funció *list()*. Aquí estem fent un canvi de tipus d'objecte de la mateixa manera que ho hem fet abans entre dades del tipus *int* i *float*.

També es pot generar una llista amb la col·lecció de valors. Ho farem de la mateixa manera que amb les claus, però en lloc de fer-ho cridant la funció *keys(),* cridarem la funció *values():*

```
valors = notes_dict.values()
valors_llista = list(valors)
```

Això ens pot servir, per exemple, per saber si una clau es troba en un diccionari o no. Per saber si hi és o no, ho farem així:

```
print('Kim' in claus_llista)
```

Aquest codi ens imprimirà *True* (si està present) o *False* (si no hi és).

I, si per alguna raó ens interessa generar una llista amb les parelles clau-valor, ho podrem fer cridant la funció *items().* El codi s'escriuria així:

```
parelles = notes_dict.items()
parelles_llista = list(parelles)
```

Per tal d'eliminar una parella clau-valor del diccionari, ho podem fer a través d'aquesta sentència:

```
del notes_dict['Joan']
```

Ara, la parella clau-valor composta per *'Joan': 4.85* ha quedat eliminada.

Un diccionari pot tenir una clau que el seu valor sigui un altre diccionari. Recorda que els valors poden ser qualsevol tipus d'objecte i els diccionaris també són un tipus d'objecte. Podem escriure aquest exemple per veure-ho més clarament:

```
nested_dict = {'a': 1, 'b': 2, 'c': {'c1': 3, 'c2': 4}, 'd': 5}
```

Una llista també pot ser un valor d'una clau d'un diccionari. Aquest seria un exemple:

```
nested_dict = {'a': 1, 'b': 2, 'c': [3,4,5,6], 'd': 7}
```

Si volem accedir i imprimir el valor a la posició 2 de la llista, que és el valor de la clau 'c', es podria fer així:

```
print(nested_dict['c'][2])
```

Primer, accedim a la llista mitjançant la clau 'c' i, després, accedim a la posició de la llista que ens interessi escrivint l'índex que correspongui.

Els diccionaris són molt útils i encara ens queden moltes coses per descobrir de Python. Seguim!

3.3 JUGAR AMB DADES

En aquest vídeo farem un exercici sobre diccionaris i llistes que et servirà per després posar en pràctica els coneixements de manera autònoma.

Comencem!

Una vegada tens PyCharm obert, pots crear un altre fitxer de Python. Recorda, botó dret a sobre la carpeta, selecciona *new* i després *Python File*.

Suposem que ha arribat l'estiu i vols saber quantes samarretes tens a l'armari en funció del color. Per això et crees un diccionari. Posarem l'exemple següent:

```
samarretes = {'vermelles': 1, 'blanques'; 5, 'blaves': 2, 'negres': 3}
```

Veus com tenim les claus del diccionari i els seus respectius valors, oi? Sí, els colors en forma d'*strings* són les claus i els números (en aquests cas, enters) són els valors de cada clau. Fins aquí bé, no?

Imagina que ha passat un temps i no trobes una samarreta de color groc. Així que vas al diccionari de samarretes i comproves si en tenies alguna de color groc. Com ho fem per saber-ho?

Bé, hem de conèixer si cap de les claus correspon a 'grogues'. Primer, doncs, generem la col·lecció de les claus i, després, transformem aquesta col·lecció en una llista.

```
claus = samarretes.keys()
print('grogues' in list(claus))
```

Fes memòria, si ens imprimeix el valor *boolean False*, voldrà dir que no tenim samarretes grogues. En canvi, si canviem 'grogues' per 'vermelles', veurem que s'imprimeix *True* i, per tant, voldrà dir que sí que tenim samarretes vermelles a l'armari.

Imagina't que, després de veure que no tens cap camiseta groga a l'armari, et proposes comprar-ne una i afegir-la al diccionari. Recordes com fer-ho?

Només hem d'escriure això:

```
camisetes['grogues'] = 1
```

Si en compres dues, escriuràs 2 en lloc de només 1.

Comprovem, ara, que efectivament tenim 'grogues' inclosa a la llista que conté les claus del diccionari:

```
claus = samarretes.keys()
print('grogues' in list(claus))
```

Fixa't que hem hagut de tornar al mateix codi una altra vegada. Quan dominis la part de funcions, no et caldrà fer-ho, però a poc a poc. Tot arriba!

Veus que en la primera comprovació s'imprimeix *False*, però en la segona obtenim *True*?

Anem a comprovar quina posició té 'grogues' a la llista de les claus? Sí va, fem-ho.

```
print(list(claus).index('grogues'))
```

Posició 4!

Anem a sumar el número de samarretes (valors) sense necessitat de fer-ho manualment d'una en una. Les llistes permeten sumar els seus valors en un sol pas:

```
valors_llista = list(samarretes.values())
print(sum(valors_llista))
```

Sí, la funció *sum()* és la que ens permet fer-ho. Així de fàcil.

Ara prova-ho tu. Segur que te'n surts.

Fins aviat!

3.4 ARA ET TOCA A TU: REPTE. LA CISTELLA DE LA COMPRA

Fixa't bé en l'enunciat i completa l'exercici següent:

Suposem que hem fet una compra en un fruiteria. Les fruites i el seu import ens apareix en el tiquet de la manera següent:

- Pomes: 3,56 €
- Mandarines: 4,35 €
- Síndria: 6,23 €
- Maduixes: 4,28 €
- Peres: 2,86
- Taronges: 3,48 €

Guarda la llista de la compra en forma de diccionari (pots ometre les unitats). Escriu el codi per tal de resoldre les qüestions següents:

- Podries calcular la mitjana de la compra accedint als valors de les claus? Procura posar només dos decimals.
- Podries copiar i guardar en una nova variable la llista dels imports sense tenir en compte els dos últims?
- Podries saber com comprovar si hem comprat llimones?

Endavant!

3.5 SOLUCIÓ EXERCICI

Com ha anat el repte de la cistella de la compra? A continuació, podrem veure els resultats pas a pas.

Primer de tot, hem de construir el nostre diccionari segons la llista de productes que tenim a la llista. Així que el diccionari s'escriuria així:

```
compra = {'pomes': 3.56, 'mandarines': 4.35, 'sindria': 6.23,
'maduixes': 4.28, 'peres': 2.86, 'taronges': 3.48}
```

Recorda, els diccionaris s'escriuen com una col·lecció de claus-valors i amb {}, a diferència de les llistes que van amb []. Les claus sempre són la part de la parella que està a l'esquerra dels : i els valors són a la dreta.

Aquest diccionari, tal com es mostra, el guardem en una variable anomenada *compra.* En aquest cas, les fruites són *strings* i els valors són *floats*. És important escriure-ho bé.

La primera qüestió que tenim per resoldre és calcular la mitjana dels imports. L'estratègia a seguir seria passar tots els valors a una llista, després sumar-la i dividir la suma pel número de valors que conté la mateixa llista. Ho fem així:

```
valors = list(compra.values())
mitjana = sum(valors)/len(valors)
```

Fixa't que hem guardat la llista dels valors en una variable que hem anomenat *valors.*

Si imprimim aquesta mitjana, veiem el resultat:

```
print('La mitjana de la compra es {:.2f}'.format(mitjana))
```

Aquí es mostra com reduir el nombre de decimals a només 2.

La segona qüestió és veure com copiar la llista de valors descartant les dues últimes dades. La solució a la qüestió es resol amb aquestes línies de codi:

```
nou_valors = valors[:-2]
print(nou_valors)
```

Recorda que una llista pot generar una còpia seva d'uns determinats índexs, fent ús dels claudàtors [index_inicial: index_final(no inclòs)]. Si l'índex inicial és omès, la posició inicial és 0 i, si l'índex final s'omet, es copiaran tots els valors fins a la posició final inclosa.

Com podem comprovar si hem comprat llimones? Per fer-ho, recorda que només ens cal mirar si les llimones estan a la llista de les claus del diccionari. El primer pas, doncs, és convertir totes les claus en una llista.

```
print('llimones' in list(compra.keys()))
```

Fixa't que aquí hem resolt la qüestió en una sola línia de codi. En els exercicis anteriors, n'havíem necessitat més d'una. Amb Python, les funcions es poden executar una dins d'una altra. A mesura que vagis assolint més experiència, aquesta habilitat et serà molt útil per escriure codi més net i més ràpid.

Si vols saber si una fruita que entres per teclat està dins de la llista, el codi només canviaria d'aquesta manera:

```
print('Entra la nova fruita: ')
fruita = input()
print(fruita in list(compra.keys()))
```

En aquest cas, l'intèrpret avaluarà si la dada que conté la variable fruita està dins de la llista. I, com que hem assignat a fruita el valor de l'entrada per teclat, estarem avaluant aquella fruita que nosaltres hem escrit.

3.6 IDEES CLAU: ESTRUCTURA DE DADES. LLISTES I DICCIONARIS

En aquest mòdul has après a treballar amb llistes i diccionaris, que són les estructures de dades més utilitzades en Python.

Les **llistes** són una col·lecció de dades del mateix tipus o no, a les quals es pot accedir mitjançant un índex de posició. Cada dada a la llista té assignat el seu propi índex. Hi ha una particularitat i és que el primer índex sempre és 0 i no pas 1. A més a més, amb les llistes, has après a fer les accions següents:

- Saber si una dada forma part de la llista mitjançant la sentència *in,* la qual torna un valor veritable o fals, depenent de si en forma part o no.
- Afegir una nova dada amb *append().*
- Treure l'última dada amb *pop().*
- Saber quina posició té una dada amb *index().*
- Sumar els valors que conté la llista amb *sum().*
- Saber el número de dades que conté una llista amb *len().*

Els **diccionaris** són una col·lecció de dades formades per parelles clau-valor. Per accedir a la informació d'una dada, cal saber la clau associada i no la seva posició, com es fa amb les llistes. Amb els diccionaris, has après a:

- Afegir una nova parella clau-valor.
- Saber quantes parelles clau-valor té el diccionari.
- Fer ús de la funció *get()* per accedir a una informació.
- Extreure una llista del conjunt de les claus.
- Generar una llista del conjunt dels valors.
- Generar una llista del conjunt de parelles clau-valor.
- Eliminar una parella clau-valor.

També hem vist que els objectes, les funcions i les estructures de dades estan connectades entre elles i no són entitats aïllades.

4 ESTRUCTURES CONDICIONALS I CICLES

En les properes activitats faràs realment un pas endavant per convertir-te en programador o programadora i aconseguiràs les habilitats que et permetran dissenyar el teu primer algoritme. Veurem les sentències condicionals i els cicles.

Les sentències condicionals són blocs de codi que ens serveixen per executar una sèrie d'accions o unes altres, depenent d'unes determinades condicions. Aquestes estructures, doncs, fan possible dissenyar algoritmes capaços de resoldre problemes molt complexos.

El cicles són blocs de codi que es repeteixen sempre que es compleixi una condició. Aquesta eina permet repetir una sèrie d'accions sense necessitat de repetir les mateixes línies de codi.

Les condicions i els cicles ens esperen. Comencem!

4.1 DOMINAR LES ESTRUCTURES CONDICIONALS. PRIMER ALGORITME

Les sentències condicionals permeten executar un codi o un altre, depenent del compliment, o no, d'una condició prèvia.

Aquí s'exposa un exemple senzill de com s'escriu una estructura condicional:

```
n = 8
if n > 10:
    print('major que 10')
elif n < 10:
    print('menor que 10')
else:
    print('igual que 10')
```

En aquest cas, estem dient a l'eina que, si la variable *n* és més gran que 10, imprimeixi '*major que 10'* i, si és més petita que 10, que imprimeixi *'menor que 10'*; si *n* no és ni menor ni major de 10, aleshores li diem que imprimeixi '*igual que 10'.*

Si ens hi fixem, podem traduir les sentències següents d'aquesta manera, partint de l'anglès:

- if n > 10 significaria 'en cas que n sigui més gran que 10'
- elif n < 10 significaria 'sinó i en cas que n sigui més petit que'
- else 'sinó…' (és a dir, qualsevol altra possibilitat)

Les sentències *if*, *elif* i *else* sempre acaben en : –dos punts– i el codi que s'executa en el cas de complir-se la condició sempre s'haurà d'indexar, com a mínim, una posició més cap a la dreta. Aquesta sintaxi és molt característica de Python. A diferència d'altres llenguatges de programació, les condicions no van entre parèntesis i els blocs de codi no s'han de posar entre '{}', només s'han d'indexar.

La sentència *if* sempre va en primer lloc.

Les sentències *elif* (una o més d'una) van en segon lloc.

La sentència *else* sempre va al final.

A continuació, introduirem els operadors comparatius, que emeten un resultat de veritat o fals, segons si es compleix la condició o no:

> (més gran que)

< (més petit que)

>= (més gran o igual que)

<= (més petit o igual que)

== (igual que)

!= (no igual que)

Si fem servir aquests operadors dins de la funció *print()*, ho veurem.

```
n = 10

print(10 < n)
```

En aquest cas, s'imprimirà *False*.

Nota:

```
if n = 10:
```

El codi de dalt donarà error. Per avaluar una condició (igual o no) s'ha de fer ús de l'operador ==.

Treballem ara amb llistes i condicionals junts:

```
noms = ['Ernest', 'Laura', 'Miquel', 'Maria']

if noms[1] == 'Laura':
    print('nom correcte')
else:
    print('nom incorrecte')
```

En aquest cas, estem comprovant si el nom de la posició 1 de la llista *noms* és 'Laura'. En cas afirmatiu, s'ha d'imprimir '*nom correcte*' i, en cas contrari, s'ha d'imprimir '*nom incorrecte'*.

Recordes la sentència *in* que fèiem servir per veure si un dada estava dins d'una llista? Doncs bé, aquesta la pots utilitzar juntament amb els condicionals. Ho farem d'aquesta manera:

```
noms = ['Ernest', 'Laura', 'Miquel', 'Maria']

if noms[1] in noms:
    print(noms[1],'està present en noms')
else:
    print(noms[1], 'no està present en noms')
```

Aquí estem fent servir la sentència *in* com a condició, el resultat de la qual determinarà si s'executa un codi o un altre. En aquest cas, com si existeix la primera posició en la llista "noms" donarà el resultat de "Ernest està present en noms".

Nota: dins de la funció *print(),* les comes serveixen per encadenar *strings*, afegint-hi un espai.

Vegem un altre exemple. En aquest cas, fent ús de números per veure operadors comparatius:

```
if n >= 10:
    print('major o igual que 10')
elif n <= 5:
    print('menor o igual que 5')
else:
    print('major que 5 i menor que 10')
```

Aquí estem avaluant si n és més gran o igual que 10, o bé si n és més petit o igual que 5. En el cas que cap de les dues condicions es compleixi, s'imprimirà 'major que 5 i menor que 10'.

Vegem una estructura condicional que doni peu a executar un codi més complex.

```
nom_nou = input()
n = int(input())

noms = ['Ernest', 'Laura', 'Miquel', 'Maria']



if n <= 20:
    noms.append(nom_nou)
else:
    noms.pop()

print(noms)
```

El que estem fent aquí és, primer, introduir per teclat un nou nom i un número. Recorda que tots els *inputs* són *strings* (per això fem ús de la funció *int()* per fer el canvi a número enter). Després, el número introduït el posem com a condició i, segons si aquest número és menor o igual que 20, s'afegirà a la llista el nou nom o, en el cas contrari, s'eliminarà el de l'última posició.

També podem fer servir els condicionals i els diccionaris junts. Aquí en tenim un exemple:

```
dades = {'nom': 'Miquel', 'ciutat': 'Barcelona', 'edat': 28}

if dades['ciutat'] == 'Barcelona':
    dades['barri'] = 'Eixample'

if dades['edat'] < 30:
    dades['categoria'] = 'Jove'

print(dades)
```

Aquí estem avaluant dos valors de dues claus diferents del diccionari. En el cas que la ciutat sigui Barcelona, s'afegirà una nova clau-valor *('barri': 'Eixample')*. A més, també s'avalua si l'edat és menor que 30 i, en el cas que sigui així, el diccionari tindrà una altra parella clau-valor *('categoria': 'Jove')*.

En aquest cas, s'imprimirà: {'nom': 'Miquel', 'ciutat': 'Barcelona', 'edat': 28, 'barri': 'Eixample', 'categoria': 'Jove'}.

4.2 REPÀS DE LES ESTRUCTURES CONDICIONALS *IF*

Hola!

Com ha anat la introducció a les estructures condicionals? Ara en farem un repàs i anirem, fins i tot, una mica més lluny. Som-hi!

Escriurem un petit programa que determini quines accions executar, segons el temps que faci. Comencem declarant dues variables (temperatura i probabilitat de pluja), que tindran les dades que entrem per teclat.

```
temp = int(input())
prob = float(input())
```

Tot seguit, afegim un diccionari que indicarà si portem jaqueta o no, quin tipus de pantalons i quin tipus de calçat. Finalment, escrivim un codi que modifiqui els valors de les claus del diccionari, segons la temperatura i la probabilitat de pluja.

```
temp = int(input())
prob = float(input())



dic = {'jaqueta': False, 'pantalons': 'curts', 'calçat': 'normal'}

if temp <= 20:
    dic['jaqueta']= True
    dic['pantalons']= 'llargs'
    if prob > 75:
        dic['calçat']='hivern']
```

T'hi has fixat? Dins del codi del primer *if*, hem posat un altre *if*. Realment no tenim cap límit per posar *ifs* uns dins dels altres! Bàsicament, el que estem fent és que una condició depengui primer d'una altra que és prèvia. En aquest cas, per exemple, la condició de mirar la probabilitat de pluja s'avalua després d'haver avaluat primer la temperatura.

```
elif temp > 20:
    if prob > 80:
        dic['calçat']= 'botes d\'aigua'
```

En el cas que la temperatura sigui superior a 20, només podrem canviar el valor de la clau 'calçat', en el cas que la probabilitat de pluja sigui més gran que 80. Veiem, doncs, que el nostre criteri per avaluar la condició de pluja varia segons una condició prèvia.

```
temp = int(input())
prob = float(input())



dic = {'jaqueta': False, 'pantalons': 'curts', 'calçat': 'normal'}

if temp <= 20:
    dic['jaqueta'] = True
    dic['pantalons'] = 'llargs'
    if prob > 75:
        dic['calçat']= 'hivern'
    print(dic)

elif temp > 20:
    if prob > 80:
        dic['calçat']= 'botes d\'aigua'
    print(dic)
```

Fixa't que en la clau 'calçat', l'apòstrof té una barra al davant. Aquesta és una sintaxi per tal que l'intèrpret identifiqui l'apòstrof com a tal i no com una cometa que marca la finalització de la cadena de caràcters.

Els condicionals es poden complicar o ser molt més complexos, però l'estructura sempre serà la mateixa. Recorda sempre el següent:

- La primera condició comença amb un *if*.
- Si no es compleix la primera i hi ha una altra condició possible *elif* (en pots posar tants com necessitis).
- Finalment, escriuràs *else* per tal d'executar aquell codi quan no es compleixi cap de les condicions prèvies.
- Les condicions no van dins de parèntesis.
- Les sentències acaben amb dos punts.
- El codi per executar dins del bloc de la condició ha d'anar indexat una posició cap a la dreta.

Ara et toca a tu! No et rendeixis. Encara hi ha molt més per aprendre.

4.3 DOMINAR ELS CICLES. SEGON ALGORITME

Els cicles són estructures que ens permeten repetir els algoritmes tantes vegades com s'especifiqui en la condició marcada, o fins que una condició determinada deixi de ser vàlida. Principalment, hi ha dos cicles: els *for* i els *while.*

Els cicles *for* tenen aquesta estructura bàsica:

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

A l'inici, sempre hem de fer servir l'expressió *for* seguida d'una nova variable (en aquest cas, l'anomenem *i*, que anirà assolint els valors que li marqui una seqüència iterable). La seqüència iterable va precedida per l'expressió *in*. Les accions que estiguin dins del bloc *for* (aquelles que estan indexades un espai a la dreta després dels dos punts) s'aniran repetint fins que *i* hagi assolit tots els valors de la seqüència iterable.

En l'exemple anterior, la funció *range()* genera una llista de valors que van des de 0 fins a 9. La variable *i*, doncs, primer agafa el valor 0 i aquest és imprès, després passa el mateix amb l'1, el 2, el 3, i així successivament fins arribar al 9 (10 – 1).

Si la funció *range(),* en lloc de tenir dins del seu parèntesi el número 10 tingués el 23, la seqüència generada aniria del 0 al 22. L'últim valor de la seqüència que genera la funció *range()* no està inclòs. De fet, la funció *range()* pot tenir un valor inicial (que sí que s'inclou en la seqüència) i que és 0 per defecte, si no s'especifica el seu valor. Aquest seria un exemple:

```
for j in range(3,10):
    print(j)
```

En aquest exemple, la seqüència comença al 3 i acaba al 9. Fixa't que ara la variable en qüestió es diu *j*. Pots posar-li el nom que vulguis.

També podem treballar amb seqüències iterables que ja tinguem en el nostre codi, com per exemple les llistes. L'estructura seria així:

```
numeros = [5,8,12,24]

for num in numeros:
    print(num)
```

La llista guardada a la variable *numeros* és iterada sencera des del primer número fins a l'últim. La nova variable anomenada *num* assolirà tots els valors (d'un en un) que hi ha a la llista i la funció *print()* els imprimirà.

Fixa't, doncs, que el que estem fent amb el cicle *for* és repetir la funció *print()* fins que hem iterat tots els valors de la seqüència donada.

Dins del bloc anterior, hem fet servir només la funció *print()*, però podem posar tantes línies de codi com vulguem. Aquí exposem algunes línies més de codi més complexes:

```
numeros = [5,8,12,24,13,4,6,11]
major10 = []
menor10 = []

for num in numeros:
    if num > 10:
        major10.append(num)
    else:
        menor10.append(num)

print('major10: ', major10)
print('menor10: ', menor10)
```

Aquest exemple ens mostra com podem afegir, de la llista *numeros*, els valors més grans que 10 en una nova llista anomenada *major10* i, els valors més petits que 10 en una altra llista anomenada *menor10.*

Els cicles *while*, en lloc de repetir codi seguint una seqüència iterable, el que necessiten és que una condició sigui verdadera per repetir el codi. Quan la condició deixa de ser verdadera, el codi deixa de repetir-se. Vegem-ne un exemple senzill:

```
n = 5
while n < 10:
    print(n)
    n = n + 1
```

En aquest cas, no hi ha cap seqüència iterable, sinó que apareix una condició que diu: *mentre* n *sigui més petit que 10*, *executa el codi*. Aquest codi imprimeix el valor actual de *n* i, tot seguit, suma una unitat a *n*. Quan *n* assoleixi el valor de 10, el codi ja no s'executarà més.

Nota: tot i que, dins del codi, la condició deixi de ser certa, primer s'executarà tot el bloc i, quan sigui el moment de començar de nou el cicle, el codi no s'executarà.

Fixa't en l'exemple següent. La suma de *n* + 1 va abans de la funció *print()*, però el resultat és el mateix. En l'últim cicle, quan *n* és igual a 10, la condició deixa de ser certa abans de la funció *print()*, però el cicle es detindrà quan s'hagi completat tot el codi del bloc. Això vol dir que la condició es posa a prova només a l'inici (a la línia del *while*) i no durant el bloc.

```
n = 5
while n < 10:
    n = n + 1
    print(n)
```

Amb els cicles *while,* podem afegir codis tan complexos com calgui. Fixa't en aquest exemple:

```
noms = ['Joan', 'Maria', 'Enric', 'Silvia', 'Aleix']
j = 0
trobat = False

while noms[j] != 'Silvia':
    print(Hem trobat la Silvia?', trobat)
    j = j + 1

trobat = True
print('Hem trobat la Silvia?', trobat)
```

Aquí anirem repetint el codi fins que trobem el nom Sílvia.

Nota: l'operador comparatiu *!=* significa NO IGUAL. Recorda que l'operador comparatiu == significa IGUAL.

4.4 REPÀS DE LES ESTRUCTURES *FOR* I *WHILE*

Hola!

Els cicles són eines molt potents que ens permeten resoldre problemes amb molta agilitat. Són estructures simples, sí, però els blocs de codi que hi contenen poden arribar a ser molt complexos. No hi ha límits. De fet, en programació no hi ha límits. Pots arribar a desenvolupar un codi molt complex, fins i tot amb els coneixements que has anat adquirint fins ara amb aquest curs.

Ara repassarem i treballarem amb les estructures *for* i *while*. Recorda que les estructures *for* recorren una seqüència iterable i les estructures *while* s'executen mentre una condició sigui certa.

Si tenim aquest exemple:

```
for i in range(2,20,2):
    print(i)
```

Aquí veiem que la funció *range()* incorpora un tercer element. Aquest tercer número marca com s'avançarà en la seqüència des del 2 fins al 19. Recorda que el 20 no hi està inclòs. Si executem el codi, veiem que no s'imprimeixen tots els números, sinó que apareixen cada dos. És justament això el que indica el tercer número. Si, per exemple, el tercer element de la funció *range()* és un 3 en lloc d'un 2, aleshores els números s'imprimiran cada tres.

I, ara, començarem des del 20 per acabar amb el 3. Com ho podem fer?

```
for i in range(20,2,-1):
    print(i)
```

Ara, el primer element és el 20 (inici), el segon és el 2 (final i no inclòs) i el tercer ens marca com avançar per la seqüència. Com que el tercer element és negatiu, el sentit serà a la inversa. I, si en lloc de tenir un -1 tenim un -2, aleshores els números apareixeran cada dos.

```
for i in range(20,2,-2):
    print(i)
```

Anem a fer una cosa divertida amb el *while*.

```
import random

n = 0
cicles = 0
while n != 5:
    print('n no és 5')
    n = random.randint(0,10)
    cicles = cicles + 1

print('Visca!!, hem trobat el 5 en el cicle', cicles)
```

Aquí, el que volem veure és quants cicles necessita el sistema per trobar aleatòriament el número 5. Fins que no el trobi, repetirà el codi, i, quan finalment el tingui, ens dirà quants cicles han calgut.

Per poder generar un element aleatòriament, hem d'importar un mòdul que es diu *random* i fer ús d'una funció que genera aleatòriament un valor enter entre 0 i 10.

Com hem dit abans, les estructures *for* i *while* són molt senzilles, però el codi que contenen en el seu bloc pot arribar a ser molt complex.

Seguim!

4.5 ARA ET TOCA A TU: REPTE. SER O NO SER

Genera una seqüència iterable formada per una llista de 10 números enters majors que 0 i menors que 20. Procura que tots els números d'aquesta llista els entris per teclat. Una vegada tinguis la llista, realitza les accions següents:

- Fes que tots els elements siguin multiplicats per 3, si són menors que 10, o per 2, si són majors que 10. Fes ús de l'estructura *for.*
- Imprimeix la llista. Fes ús de l'estructura *for.*
- A continuació, imprimeix només els cinc primers números. Fes ús de l'estructura *while*.

Si vols, i de manera opcional, intenta fer aquest últim pas:

Torna a fer les accions anteriors, però en lloc d'introduir els números per teclat, fes-ho important el mòdul *random* i fent ús de la funció randint().

4.6 SOLUCIÓ EXERCICI

Has aconseguit resoldre l'exercici? Felicitats! Si no ha estat així, no passa res. Si és la primera vegada que programes, és molt normal que et resulti una mica complex. La clau per arribar a programar és practicar molt i reduir els problemes complexos a una col·lecció de problemes més petits i senzills. Aquí tens el codi (tot i que segurament hi haurà més opcions, com passa sempre en programació).

A continuació, l'explicarem pas a pas:

```
print('Introdueix 10 numeros')
llista = []
for i in range(0,10):
    print('imprimeix posició:', i)
    llista.append(int(input()))

print(llista)

for j in range(0, len(llista)):
    if llista[j] > 10:
        llista[j] = llista[j] * 2
    if llista[j] < 10:
        llista[j] = llista[j] * 3

print(llista)

j = 0
while j < 5:
    print(llista[j])
    j = j + 1
```

Primer, generem una llista buida i l'anem omplint amb els valors que entrem per teclat. Fixa't que aquí tenim el primer cicle *for*. Com que necessitem 10 valors, el que fem és fer un cicle amb una seqüència iterable generada per la funció *range().* En cada cicle, el sistema ens indica que hem d'introduir un valor per teclat a la posició corresponent. Recorda que les entrades són *strings* i que necessitem canviar-les a nombres enters. Per això, fem servir la funció *int().* Una vegada tenim la llista plena, diem a l'eina que l'imprimeixi. Bé, ja tenim una part de l'exercici resolt.

La segona part ens diu que hem de canviar els valors dels elements de la llista. Si són més grans que 10 els hem de multiplicar per 2 i, si són més petits, per 3. Aquí, la clau és saber que, per canviar el valor d'una llista, necessitem accedir-hi, i això es fa mitjançant la seva posició. Per tant, hem de generar una seqüència iterable que correspongui a les posicions de la llista. La funció *range(),* doncs, l'escriurem de tal manera que el primer valor sigui 0 i l'últim sigui 9. Pots escriure aquest valor final a mà, posant explícitament 10 (recorda que l'última posició, doncs, seria 10 – 1) o, com en el codi que mostrem, fent servir la funció *len(),* que també ens tornarà el valor 10. A partir d'aquí, cada valor de la seqüència és avaluat i multiplicat per 3 o per 2, depenent de si és més gran o més petit que 10. Si el número fos 10, no faria res. Imprimim la llista, per comprovar si el nostre codi canvia els seus valors, segons els requisits del problema.

Per al tercer pas, que ens diu que només hem d'imprimir els cinc primers valors, necessitem que la condició que acompanya a *while* posi fi al cicle quan j sigui igual a 5. Mentre que j (en aquest cas correspon a la posició de la llista) sigui inferior a 5, el cicle tornarà a començar.

Nota: és molt important que al final del bloc sumem a j una unitat (j = j + 1). Si això no ho fem, el cicle es repetirà eternament, ja que la condició per posar fi al cicle sempre serà verdadera i, per tant, mai s'aturarà.

Per resoldre l'exercici generant la llista de 10 números de manera aleatòria en lloc d'introduir els valors a mà, el codi canviaria d'aquesta manera. Fixa't que el canvi és realment de només dues línies:

```
import random

print('Introdueix 10 numeros')
llista = []
for i in range(0,10):
    llista.append(random.randint(1,19))

print(llista)

for j in range(0, len(llista)):
    if llista[j] > 10:
        llista[j] = llista[j] * 2
    if llista[j] < 10:
        llista[j] = llista[j] * 3

print(llista)

j = 0
while j < 5:
    print(llista[j])
    j = j + 1
```

El que hem fet és afegir la primera línia de codi per importar el mòdul *random*. Després, només hem necessitat canviar el que hi ha dins dels parèntesis de la funció *append()*, introduint la funció *randint().*

4.7 IDEES CLAU: ESTRUCTURES CONDICIONALS I CICLES

En aquest mòdul hem après a fer ús dels **cicles *for*** i ***while***. Per una banda, els cicles *for* ens permeten repetir blocs de codi a través d'una seqüència iterable. Per altra banda, podem executar el codi que hi ha a dins de l'estructura *while* repetidament, sempre que la condició que marquem sigui certa.

Els blocs de codi, tant per als cicles *for* com per als cicles *while*, són aquelles línies de codi que van després dels : –dos punts– i que estan indexades una posició a la dreta. De fet, qualsevol bloc de codi en Python que formi part d'una estructura sempre s'escriurà així.

La funció *range()* ens permet generar una seqüència iterable per ser utilitzada en els cicles *for*. El primer valor marca l'inici de la seqüència, el segon marca el final (el valor no està inclòs) i el tercer marca l'avanç a través d'aquesta seqüència.

La sentència *in* és la que ens indica sobre quina seqüència s'executarà el cicle *for*. De fet, l'expressió *in* també pot referir-se directament a una llista i tractar-la com una seqüència iterable.

També hem après a generar una seqüència de valors enters de manera aleatòria fent ús de la funció *randint()* del mòdul *random*.

La funció *while* avalua la condició només a l'inici del cicle. Això vol dir que, si durant el bloc de codi no es compleix la condició, primer s'executarà tot el bloc i s'aturarà la repetició només quan el cicle torni a començar.

5 CONSTRUCCIÓ DE FUNCIONS

Les funcions permeten replicar blocs de codi allà on els necessitem. Això vol dir que només les haurem de definir una vegada per fer-les servir després sense límit de rèpliques. Les funcions poden contenir, dins dels seus blocs, totes les sentències i estructures que hem après durant el curs.

Les funcions poden ser simples o molt complexes i, fins i tot, es poden fer servir una dins d'una altra. Imagina que vols crear un joc en el qual el protagonista és un avatar que ha de superar diverses aventures. Totes les accions que podrà fer aquest avatar estaran definides per les funcions que tu defineixis. Saltar, anar a l'esquerra, atacar... tot això són funcions.

Les funcions es poden emmagatzemar en una variable. Sí, i també són objectes. Recorda: a Python, tot són objectes.

5.1 APRENDRE A CONSTRUIR FUNCIONS I COM INVOCAR-LES

Una funció és un bloc de codi reutilitzable. La seva missió principal és generar una acció o un canvi i que aquest es pugui dur a terme només invocant la funció sense tenir la necessitat de tornar escriure el codi. Aquest és un exemple senzill:

```
def duplicar(x):
    y = x * 2
    print (y)
```

Aquesta funció s'encarrega d'imprimir el doble de *x*. Analitzem la composició d'una funció:

- La sentència *def* defineix que l'objecte és una funció.
- *duplicar* és el nom de la funció, tot i que podria ser qualsevol altre.
- (x) és un paràmetre que llegeix la funció i després el fa servir en el codi.
- *print()* és l'acció que en aquest cas fa la funció.

Sí, una funció pot estar dins d'una altra funció. En aquest cas, la funció *print()* ja ve configurada amb el paquet Python i no cal tornar a picar tot el codi. Només cal invocar-la dins de la nova funció, en aquest cas, la funció *duplicar*.

Cal dir, però, que la funció, de moment, no genera cap canvi ni executa res. Per tal que s'executi el codi de la funció, cal invocar-la. Per fer-ho, només caldrà escriure:

```
duplicar(5)
```

Ara, si veus el resultat a la terminal, el resultat que apareix és 10. Quan una funció s'invoca, s'han d'assignar als paràmetres els seus valors, per tal que es facin servir durant l'execució del codi.

Veuràs que el codi de la funció s'empaqueta de la mateixa manera que els condicionals o els cicles. Es posen els dos punts (:) a la primera línia i, tot seguit, el codi s'indexa una posició a la dreta. El codi que pot contenir una funció pot ser tan complex com calgui. Anem a escriure quelcom més interessant.

Nota: tot allò que ja no estigui indexat dins la funció, no en formarà part. El condicional *while* no és part de la funció encadenar_paraules.

```
majors_cinc = []
menors_igual_cinc = []
repliques = 0


def encadenar_paraules(num):
    if num > 5:
        majors_cinc.append(num)
    else:
        menors_igual_cinc.append(num)


while repliques < 5:
    print('introdueix número')
    encadenar_paraules(int(input()))
    repliques += 1


print('número rèpliques és', str(repliques))
print('numeros majors 5', majors_cinc)
```

Aquest exemple mostra com n'és d'útil fer servir funcions. La funció *encadenar_paraules()* afegeix el valor del seu paràmetre a la llista *majors_cinc* o *menors_igual_cinc*, depenent de si aquest és major que 5 o no. Per tal de no haver de repetir els condicionals i les funcions *append()* durant el cicle *while*, dissenyem primer la funció i després només cal invocar-la dins del bloc *while*. Aquesta funció serà invocada tantes vegades com cicles es facin fins que la condició de *while* deixi de complir-se. Fixa't que el valor del paràmetre *num* assoleix el valor que introduïm per teclat a través de *input().*

Tot seguit, redactarem una funció que sigui capaç de tornar un valor. Això voldrà dir que, cada vegada que executem la funció, aquesta podrà emetre un resultat. La manera que té una funció de generar un resultat és mitjançant la sentència *return*. Vegem un exemple senzill:

```
def duplicar(x):
    return x * 2

y = duplicar(3)

print(y)
```

Hem fet servir la mateixa funció que a l'inici. Ara, però, la funció genera un resultat (*x* multiplicat per 2), que es pot assignar després a una variable. Aquí hem escollit una variable *y*. Per tant, cada vegada que cridem o invoquem la funció, assignant al seu paràmetre *x* un valor, generarem un resultat exportable a través de la sentència *return*.

La invocació d'una funció amb sentència *return* es pot fer servir en qualsevol altra operació i el valor o informació que entrarà en joc serà, de fet, aquell valor que generi aquesta operació. Per exemple, podem modificar el codi anterior d'aquesta manera:

```
def duplicar(x):
    return x * 2

y = 5
z = y + duplicar(3)
print(z)
```

Ara, la variable *y* té assignat el valor 5 (número enter). La variable *z* és la suma de la variable *y*, és a dir 5, i la del valor que genera la funció duplicar, és a dir 6. El resultat de l'operació, doncs, és 5 + 6. Aquest resultat es guarda a la variable anomenada *z*.

Les funcions poden generar qualsevol tipus d'objecte.

Per exemple, també podem generar una llista. Aquest exemple és força il·lustratiu:

```
a = [4,2,7]
b = [6,3,12]

def llista_doble(llista):
    nova_llista = []
    for i in range(len(llista)):
        nova_llista.append(llista[i] * 2)

    return nova_llista


print(llista_doble(a))
print(llista_doble(b))
```

Fixa't que el paràmetre que agafa la funció és una llista. Podem invocar la funció, per exemple, amb una llista que anomenem *a* i amb una altra que anomenem *b*. Podràs reutilitzar la funció tantes vegades com vulguis sense necessitat de canviar el codi.

5.2 COM CONSTRUIR UNA FUNCIÓ I COM TORNAR UN BON RESULTAT

Hola de nou!

A l'última part de l'article anterior has après a invocar funcions, assignant valor al seu paràmetre. Ara construirem una funció amb més d'un paràmetre, tot i que no té gaire complicació, ja que es fa de la mateixa manera que si en tinguéssim només un. De fet, una funció pot agafar tants paràmetres com es necessitin.

```
def duplicar_coordenades(x,y,z):
    j = x
    x = y
    y = z
    z = j

    print('x:', x, 'y:', y, 'z:', j)

duplicar_coordenades(4,7,9)
```

Aquí, bàsicament, hem fet un intercanvi entre tres paràmetres. És a dir, quan s'invoca una funció, els paràmetres x,y,z tenen els valors 4, 7 i 9 respectivament i, després, aquest són intercanviats segons el codi.

Ara, anem a fer una cosa més divertida i que s'apropa més a un desenvolupament de codi d'un projecte real. Ara farem que el retorn d'una funció sigui el valor del paràmetre d'una altra funció:

```
def operacio_1(x):
    return x*3 + 6

print(operacio_1(6))

def operacio_2(x):
    print(x/3 - 6)

print(operacio_2(operacio_1(6)))
```

Potser veus el codi una mica complex, si mai no has programat. Si és així, vol dir que estàs davant d'una cosa nova i que només et cal una mica de temps. Repassem el codi que hem escrit.

La primera part, la funció *operacio_1*, és senzilla. És una funció que genera un valor que és el triple del valor que assoleix el paràmetre *x* més 6. Si invoquem la funció operacio_1 dins de la funció *print()* i a *x* li donem el valor de 6, el resultat que obtindrem serà 24.

Després, definim la funció *operacio_2*, que tampoc no té gaire complicació. El que fa és, d'un valor *x*, tornar un altre valor que és el resultat de dividir per 3 i restar-li 6. Ara, és aquí quan arriba la part més crítica.

Quan invoquem la funció operacio_2 dins de la funció *print(),* el valor del paràmetre serà 24, que és el resultat de la funció *operacio_1*. La invocació de la funció assolirà el valor que es generi amb la sentència *return* i, per tant, aquest valor podrà ser assignat al paràmetre de la funció *operacio_2*.

Recorda que, en aquest cas, la funció *print()* imprimeix un número amb decimals, perquè és el resultat d'una divisió. Si, per alguna raó, volguéssim que fos un número enter, hauríem de substituir la barra diagonal per una de doble. El codi seria així:

```
print(x//3 - 6)
```

Abans d'acabar aquest vídeo, podríem fer una última cosa. Recorda que a Python tot són objectes i, per tant, les funcions també. Això vol dir que podem guardar la funció sencera en una variable (i no només el resultat que hi genera).

```
def operacio_1(x):
    return x*3 + 6

f = operacio_1

print(f(4))
```

Ara la funció operacio_1 queda guardada en una variable *f* i podem invocar-la a través de la seva variable i assignar el valor del paràmetre de la mateixa manera.

Ara sí que has fet un pas endavant!

Continuem.

5.3 ARA ET TOCA A TU: REPTE. FES-HO UNA VEGADA I CRIDA-LA TANTES VEGADES COM VULGUIS

A continuació, et proposem resoldre un exercici vinculat a les funcions.

Suposem que tenim una col·lecció de números enters guardats en una llista. Construeix una funció que agafi com a paràmetre aquesta llista i que torni una nova llista els valors de la qual siguin el doble de la llista d'entrada.

Per exemple, si el paràmetre de la funció és [4,6,8], el retorn serà [8,12,16].

Construeix una nova funció (que anomenarem funció_1) que agafi com a paràmetre i que, primer, multipliqui per 3 els seus valors, si aquests no són superiors a 10 i, després, que imprimeixi la llista. Invoca aquesta funció de tal manera que el valor del seu paràmetre sigui la llista quetorni la primera funció.

Resol l'exercici per aquestes tres llistes diferents que agafen la funcio_1 com a paràmetre:

[4,5,7]

[1,3,2]

[3,6,8]

Procura resoldre l'exercici suposant que la llista que agafa com a paràmetre la funcio_1 és una seqüencia de quatre valors enters generats aleatòriament de l'1 al 9.

5.5 SOLUCIÓ EXERCICI

Abans de solucionar l'exercici, és important llegir l'enunciat i analitzat quines eines de codi ens seran útils. En aquest cas, per exemple, i una vegada haguem vist que hem de transformar una llista, el més habitual és pensar que els cicles *for* ens seran de gran ajuda. Aquí es mostra el codi que soluciona l'exercici per a les tres llistes donades [4,5,7], [1,3,2] i [3,6,8]:

```
seq = [4,5,7]

def funcio_1(llista):
    for i in range(len(llista)):
        llista[i] = llista[i] * 2

    return llista


def funcio_2(llista):
    for i in range(len(llista)):
        if llista[i] <= 10:
            llista[i] = llista[i] * 3
    print(llista)


funcio_2(funcio_1(seq))
```

Sempre que necessitem recórrer una seqüència i canviar els seus valors, haurem d'accedir a totes les seves posicions. I, una vegada estiguem a la posició correcta, podrem fer el canvi de valor.

La *funcio_1* recorre tota la llista i multiplica per 2 tots els seus valors. I, després, fem servir la sentència *return* per tal que la funció generi com a resultat la nova llista, quan sigui invocada.

La *funcio_2* fa exactament el mateix que la *funció_1*, però, en lloc de multiplicar per 2, ho fa per 3, tot i que abans imposa una condició: en aquells números de la llista d'entrada que siguin superiors a 10 no es durà a terme la seva multiplicació per 3.

Com que l'exercici ens obliga a què el valor del paràmetre de la *funció_2* sigui el valor que genera la *funció_1*, haurem d'invocar la funció_1 dins dels parèntesis de la *funcio_2*. També podríem guardar prèviament la llista que genera la funcio_1 en una variable i passar-la com a valor al paràmetre.

Per tal de resoldre la segona part de l'exercici, aquella que ens demana que el paràmetre de la *funció_1* sigui una seqüència de quatre valors enters de 1 a 9 generats aleatòriament, necessitem aquest codi:

```
import random


def funcio_1(llista):
    for i in range(len(llista)):
        llista[i] = llista[i] * 2

    return llista


def funcio_2(llista):
    for i in range(len(llista)):
        if llista[i] <= 10:
            llista[i] = llista[i] * 3
    print(llista)


def llista_aleatoria(n):
    llista = []
    for i in range(n):
        llista.append(random.randint(1,9))
    print('llista aleatoria: ', llista)
    return llista


funcio_2(funcio_1(llista_aletoria(4)))
```

La primera diferència que veiem és que necessitem importar el mòdul *random* per poder fer servir la funció *randint(),* que ens permetrà generar els valors aleatoris. I, per tal de generar una llista amb aquests valors aleatoris, necessitem escriure una funció que generi aquesta llista i que agafi un paràmetre que ens indiqui el nombre de valors (en aquest cas, l'exercici ens demana quatre valors). En el codi de dalt, aquesta funció *és llista_aleatoria.* Fixa't que, per generar la llista, necessitem, primer, declarar-ne una que estigui buida i, després, anar afegint els valors fins a tenir-los tots quatre.

L'última línia del codi s'encarrega d'invocar la funció_2, el paràmetre de la qual és la llista que genera la funció_1. Fixa't, doncs, que el paràmetre de la funcio_1 és la llista de valors aleatoris que ha generat la funció *llista_aleatoria*.

Durant la redacció de codi professional, és molt normal trobar aquestes invocacions unes a sobre de les altres. És qüestió de pràctica acostumar-se a treballar així.

5.6 IDEES CLAU: CONSTRUCCIÓ DE FUNCIONS

Ja hem acabat el curs. Repassem les idees més importants que han aparegut relacionades amb les **funcions.**

Per tal de construir funcions, primer necessitem escriure el bloc i després invocar-les per fer-les servir.

La construcció es compon de la **sentència *def***, el **nom** de la funció, els **paràmetres** i el **codi**. El codi que forma part d'aquesta funció sempre ha d'anar després de dos punts (:) i estar indexat una posició a la dreta.

Les funcions poden generar valors mitjançant la sentència ***return***. Això vol dir que una funció, quan sigui invocada, enviarà una dada que serà allò que torni la sentència *return*. Aquesta dada es pot guardar en una variable, per fer-la servir després, fer-la servir directament dins d'una altra operació o com a paràmetre d'una altra funció. De fet, les funcions poden ser invocades unes sobre les altres, fent-se passar per paràmetres.

Les funcions senceres poden ser guardades en una variable. Aquesta es pot invocar amb els paràmetres que corresponguin, per tal de dur a terme les accions de la funció.

Descobreix tot el que Barcelona Activa pot fer per a tu

Acompanyament durant tot el procés de recerca de feina

barcelonactiva.cat/treball

Suport per posar en marxa la teva idea de negoci

barcelonactiva.cat/emprenedoria

Serveis a les empreses i iniciatives socioempresarials

barcelonactiva.cat/empreses

Formació tecnològica i gratuïta per a la ciutadania

barcelonactiva.cat/cibernarium

Xarxa d'equipaments de Barcelona Activa

1. Seu Central Barcelona Activa
   Porta 22
   Centre per a la Iniciativa Emprenedora Glòries
   Incubadora Glòries
2. Convent de Sant Agustí
3. Ca n'Andalet
4. Oficina d'Atenció a les Empreses
   Cibernàrium
   Incubadora MediaTIC
5. Incubadora Almogàvers
6. Parc Tecnològic
7. Nou Barris Activa
8. innoBA

○ Punts d'atenció a la ciutat

Barcelona Activa

Ajuntament de Barcelona

Darrera actualització 2022

Cofinançat per:

UNIÓ EUROPEA
Fons Europeu de Desenvolupament Regional

Segueix-nos a les xarxes socials:

barcelonactiva.cat/cibernarium
barcelonactiva
barcelonactiva
company/barcelona-activa