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.

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

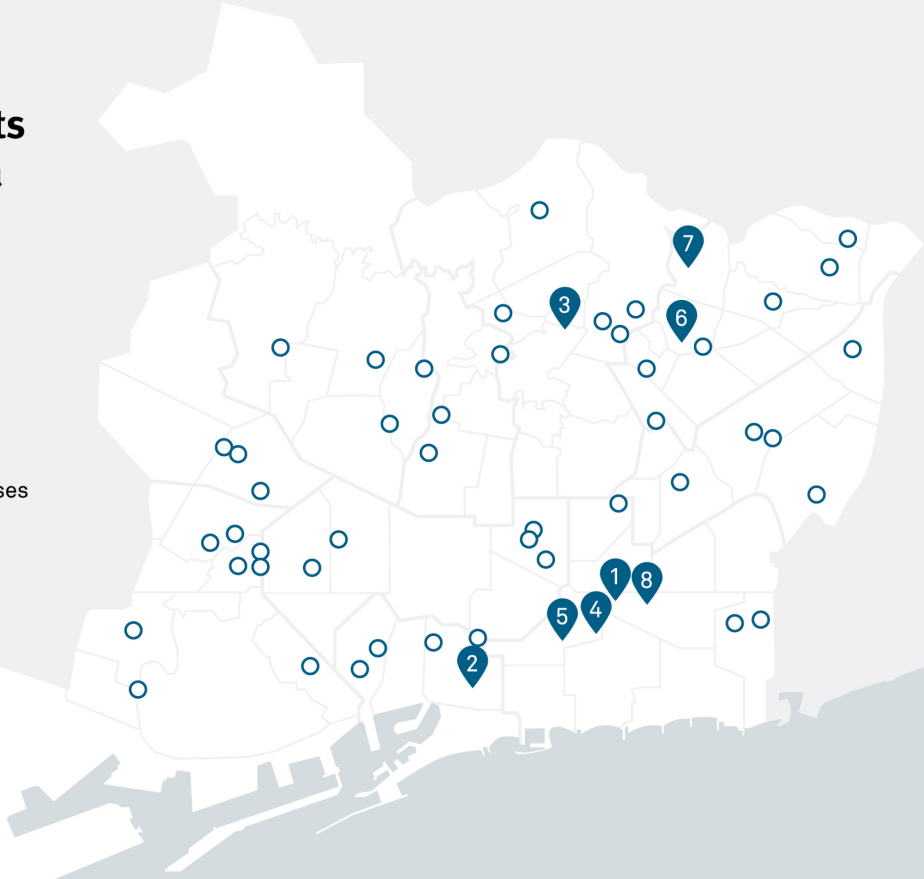
Formació tecnològica i gratuïta per a la ciutadania

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 2019

Cofinançat per:

UNIÓ EUROPEA
Fons Europeu de Desenvolupament Regional

Segueix-nos a les xarxes socials:

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