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.

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

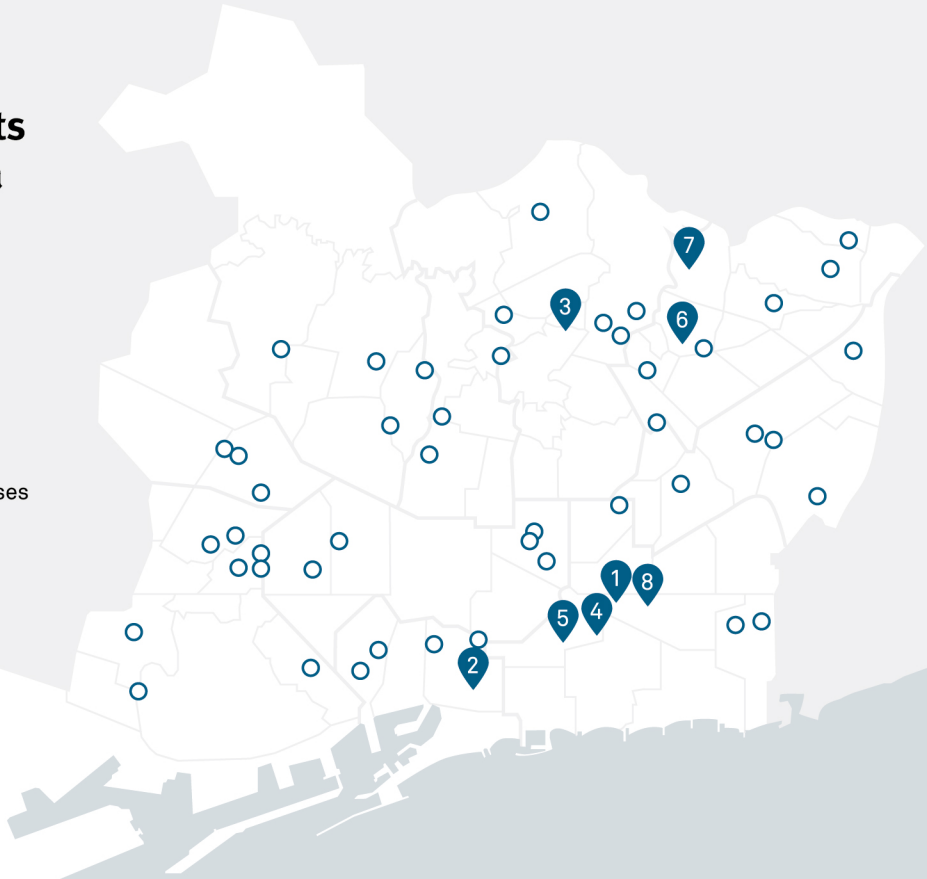
Formació tecnològica i gratuïta per a la ciutadania

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