
Python

Índex

1. Estructura d'un programa informàtic. Tipus de dades simples	1
1.1. Començant a programar en Python	1
Per què Python?	1
Instal·lació i verificació	1
L'entorn interpretat (REPL)	1
Editors i IDEs recomanats	2
Tipus de dades bàsics	2
Operadors aritmètics	3
1.2. Variables i constants	3
Variables	3
Constants	4
Conversió de tipus (<i>casting</i>)	4
1.3. Primers programes: entrada i sortida de dades	4
Sortida: <code>print()</code>	4
Formatació de cadenes (f-strings)	4
Entrada: <code>input()</code>	5
Primer programa complet	5
1.4. Metodologia	5
Joc de proves	5
Metodologia de resolució de problemes	5
2. Estructures condicionals	6
2.1. Expressions booleanes	6
Operadors de comparació	6
Operadors lògics	6
2.2. Estructures condicionals	7
if simple	7
if...else	7
if...elif...else	7
Condicionals imbricats	7
Exemple pràctic: classificador de serveis	8
La instrucció <code>match</code> (Python 3.10+)	8
3. Estructures repetitives: bucles	10
3.1. Estructura repetitiva <code>while</code>	10
Bucle amb entrada d'usuari	10
3.2. Ús avançat de <code>while</code>	11
<code>break</code> , <code>continue</code> i <code>else</code> en bucles	11
La clàusula <code>else</code> en bucles	11
La instrucció <code>pass</code>	12
Redirecció de fitxers per <code>stdin</code>	12
3.3. Estructura repetitiva <code>for in</code>	12
<code>range()</code>	13
<code>enumerate()</code> --- índex i valor alhora	13
Exemple: escaneig de ports simulat	13
3.4. Estructura repetitiva i programació estructurada	14
Bucles imbricats	14
Exemple: generador d'adreces IP	14

4. Tipus de dades compostes	14
4.1. Cadenes de text (<code>str</code>)	14
Mètodes de cadenes útils per a <code>sysadmin</code>	15
4.2. Ús avançat de cadenes	15
Mètodes addicionals	15
Formatació avançada	15
4.3. Llistes	16
Operacions comunes	16
List comprehensions	17
4.4. Llistes i cadenes. Llistes multidimensionals. Tuples i Conjunts	18
Convertir entre llistes i cadenes	18
Llistes multidimensionals (matrius)	18
Tuples	18
Conjunts (<code>set</code>)	19
5. Coneixements bàsics de funcions	19
5.1. Introducció a les funcions	19
5.2. Primers programes amb funcions	20
Funcions amb paràmetres	20
Funcions que retornen valors	20
Paràmetres per defecte	20
Anotacions de tipus (<i>type hints</i> , Python 3.5+)	20
f-strings avançades (Python 3.8+)	21
5.3. Conceptes avançats de la programació modular	21
Àmbit de variables (<i>scope</i>)	21
Funcions que criden altres funcions	21
6. Mòduls i llistes de funcions	22
6.1. Introducció als mòduls	22
Importar mòduls de la biblioteca estàndard	22
Mòduls útils per a <code>sysadmin</code>	22
Importació selectiva	23
6.2. Crear el nostre propi mòdul	23
6.3. Dunders i conceptes avançats	24
<code>__name__</code> i el bloc principal	24
<code>__doc__</code> (docstrings)	24
7. Ús de funcions per la gestió de dades compostes	25
7.1. Llistes i algorismes d'ordenació	25
Ordenació amb <code>sort()</code> i <code>sorted()</code>	25
Ordenació per criteri personalitzat (<code>key=</code>)	25
Algorisme de cerca lineal	25
7.2. Llistes de llistes	26
7.3. Diccionaris	26
Mètodes principals	27
7.4. Ús avançat de diccionaris	27
Diccionari de diccionaris (inventari de xarxa)	27
Comptar elements amb diccionaris	28
8. Ús de fitxers a Python per l'administrador de sistemes	29
8.1. Fluxos de fitxers, lectura i escriptura	29

Obrir un fitxer: <code>open()</code>	29
Forma recomanada: <code>with</code>	29
Llegir fitxers	29
Escriure fitxers	30
Exemple complet: analitzador de logs	30
Gestió d'excepcions en fitxers	31
Escriure informes amb <code>print()</code> redirigit a fitxer	32
Exemple final: script d'inventari complet	32
Guardar dades estructurades amb JSON	34
Resum de conceptes clau	35
Recursos addicionals	35
Índex de figures	36

Cicle formatiu: Administració de Sistemes Informàtics en Xarxa (ASIX)

Versió Python: 3.10+ (referència oficial: docs.python.org/3)



Figura 1: Python logo

1. Estructura d'un programa informàtic. Tipus de dades simples

1.1. Començant a programar en Python

Per què Python?

Python és un dels llenguatges de programació més utilitzats al món, especialment en l'àmbit de l'administració de sistemes. Algunes raons:

- **Sintaxi llegible i clara:** el codi s'assembla molt al pseudocodi.
- **Multiplataforma:** funciona a Linux, Windows i macOS.
- **Gran comunitat i llibries:** hi ha mòduls per gairebé qualsevol tasca d'administració (xarxa, sistemes de fitxers, seguretat...).
- **Interpretació interactiva:** pots provar codi línia a línia sense compilar.
- **Molt usat en automatització i scripting** de sistemes Linux.

Instal·lació i verificació

```
# Verificar versió a Linux
python3 --version

# Instal·lar si cal (Debian/Ubuntu)
sudo apt install python3 python3-pip
```

L'entorn interpretat (REPL)

Python disposa d'un intèrpret interactiu anomenat **REPL** (*Read-Eval-Print Loop*). S'inicia escrivint `python3` al terminal:

```
$ python3
>>> 2 + 3
5
```

```
>>> print("Hola, món!")
Hola, món!
>>> exit()
```

NOTA

El símbol >>> indica que estem dins de l'interpret. No cal escriure'l als scripts.

Editors i IDEs recomanats

Per a principiants

- **Thonny** --- dissenyat específicament per aprendre Python. Inclou un depurador visual molt clar, mostra els valors de les variables pas a pas, i no requereix configuració. Ideal per a les primeres setmanes.
- **IDLE** --- ve inclòs amb Python. Molt simple, suficient per als primers programes, però limitat per a projectes més grans.

Per a ús professional

- **Visual Studio Code** --- l'opció més recomanada per a ASIX. Lleuger, gratuït, amb l'extensió oficial de Python ofereix autocompletat, depurador integrat, terminal incorporat i suport per a scripts de sistemes. És el que més s'utilitza professionalment en entorns Linux.
- **PyCharm Community** --- IDE dedicat exclusivament a Python, molt potent. La versió Community és gratuïta. Ideal per a projectes més grans, però consumeix més recursos que VS Code.

Per a scripting ràpid des del terminal (molt útil per a sysadmin)

- **nano / vim / neovim** --- per editar scripts directament al servidor sense entorn gràfic. Com a administradors de sistemes, haureu de ser capaços d'editar fitxers .py des del terminal SSH.

Recomanació pràctica per al curs

Thonny per a les primeres unitats, VS Code a partir de la meitat del curs. Així els alumnes s'acostumen a un entorn professional sense sobrecarregar-se al principi.

Tipus de dades bàsics

Tipus	Nom Python	Exemple
Nombre enter	int	42, -7, 0
Nombre decimal	float	3.14, -0.5
Cadena de text	str	"Hola", 'Linux'
Booleà	bool	True, False

```
>>> type(42)
<class 'int'>
>>> type(3.14)
```

```
<class 'float'>
>>> type("Hola")
<class 'str'>
>>> type(True)
<class 'bool'>
```

Operadors aritmètics

Operador	Significat	Exemple	Resultat
+	Suma	5 + 3	8
-	Resta	5 - 3	2
*	Multiplicació	5 * 3	15
/	Divisió (real)	5 / 2	2.5
//	Divisió entera	5 // 2	2
%	Mòdul (resta)	5 % 2	1
**	Potència	2 ** 8	256

```
>>> 2 ** 10      # molt útil per calcular mides en bytes/KB/MB
1024
>>> 1024 * 1024  # 1 MB en bytes
1048576
```

1.2. Variables i constants

Variables

Una **variable** és un nom que fa referència a un valor emmagatzemat a la memòria. A Python no cal declarar el tipus; s'infereix automàticament.

```
# Assignació de variables
nom = "servidor01"
port = 22
ip = "192.168.1.10"
actiu = True

# Mostrar valors
print(nom)
print(port)
```

Regles per als noms de variables:

- Poden contenir lletres, dígitos i guions baixos _
- No poden començar per un dígit
- Distingeixen majúscules i minúscules (Port ≠ port)
- No poden ser paraules reservades (if, for, while...)

Constants

Python no té constants pròpies del llenguatge, però per convenció s'escriuen en **MAJÚSCULES**:

```
MAX_INTENTS = 3
PORT_SSH = 22
RUTA_LOG = "/var/log/syslog"
```

Conversió de tipus (*casting*)

```
edat = "25"          # és una cadena
edat_int = int(edat) # convertim a enter
print(edat_int + 1)  # 26

preu = 19.99
print(int(preu))      # 19 (trunca, no arrodoneix)
print(str(preu))      # "19.99"
```

1.3. Primers programes: entrada i sortida de dades

Sortida: `print()`

```
print("Hola, món!")
print("El servidor és:", "web01")
print("Port:", 80)

# Separador personalitzat
print("192", "168", "1", "1", sep=".") # 192.168.1.1

# Sense salt de línia al final
print("Carregant", end="")
print("... OK")
```

Formatació de cadenes (f-strings)

```
nom = "router01"
ip = "10.0.0.1"
print(f"Dispositiu: {nom} - IP: {ip}")
# Sortida: Dispositiu: router01 - IP: 10.0.0.1

espai_lliure = 15.7
print(f"Espai lliure: {espai_lliure:.1f} GB")
# Sortida: Espai lliure: 15.7 GB
```

Entrada: input()

La funció input() sempre retorna una **cadena de text** (str). Si necessitem un nombre, cal convertir-lo.

```
nom = input("Introdueix el nom del servidor: ")
print(f"Servidor seleccionat: {nom}")

port = int(input("Introdueix el port: "))
print(f"Connectant al port {port}...")
```

Primer programa complet

```
# programa: info_servidor.py
# Demana informació d'un servidor i la mostra per pantalla

nom = input("Nom del servidor: ")
ip = input("Adreça IP: ")
port = int(input("Port SSH: "))

print()
print("=== Informació del servidor ===")
print(f"  Nom   : {nom}")
print(f"  IP    : {ip}")
print(f"  Port  : {port}")
```

1.4. Metodologia

Joc de proves

Abans de programar, definim els casos que ha de funcionar correctament el programa:

Entrada	Sortida esperada
nom="web01", ip="10.0.0.5", port=80	Mostra correctament les dades
port="abc"	Error de conversió (cas d'error)

Metodologia de resolució de problemes

1. **Comprendre el problema:** llegir l'enunciat fins a entendre'l completament.
2. **Definir entrades i sortides:** quines dades entren i quines han de sortir.
3. **Dissenyar l'algorisme:** en pseudocodi o diagrama de flux.
4. **Codificar:** traduir l'algorisme a Python.
5. **Provar:** executar amb el joc de proves definit.
6. **Depurar:** si hi ha errors, localitzar-los i corregir-los.

CONSELL

Comença sempre pels casos més simples. Afegeix complexitat progressivament.

2. Estructures condicionals

2.1. Expressions booleanes

Una **expressió booleana** és qualsevol expressió que s'avalua com a True o False.

Operadors de comparació

Operador	Significat	Exemple	Resultat
==	Igual	5 == 5	True
!=	Diferent	5 != 3	True
>	Major que	5 > 3	True
<	Menor que	5 < 3	False
>=	Major o igual	5 >= 5	True
<=	Menor o igual	3 <= 5	True

Operadors lògics

Operador	Significat	Exemple
and	I lògic (tots dos vertaders)	a > 0 and a < 100
or	O lògic (almenys un vertader)	port == 80 or port == 443
not	Negació	not actiu

```
port = 443
protocol = "https"

# Combinació d'operadors
if port == 443 and protocol == "https":
    print("Connexió segura")

# Comprovar rang
usuari = 45
if usuari >= 10 and usuari <= 100:
    print("Nombre d'usuari vàlid")
```

2.2. Estructures condicionals

if simple

```
espai = 5 # GB lliures

if espai < 10:
    print("AVÍS: Espai en disc baix")
```

if...else

```
port = int(input("Port: "))

if port < 1024:
    print("Port reservat (requereix root)")
else:
    print("Port d'usuari")
```

if...elif...else

```
codi_resposta = 404

if codi_resposta == 200:
    print("OK - Petició correcta")
elif codi_resposta == 301:
    print("Redirecció permanent")
elif codi_resposta == 403:
    print("Accés prohibit")
elif codi_resposta == 404:
    print("Recurs no trobat")
elif codi_resposta == 500:
    print("Error intern del servidor")
else:
    print(f"Codi desconegut: {codi_resposta}")
```

Condicionals imbricats

```
ip = input("Adreça IP: ")
port = int(input("Port: "))

if ip.startswith("192.168."):
    print("Xarxa local detectada")
    if port == 22:
        print(" Accés SSH permès")
    elif port == 3306:
        print(" Accés a MySQL (revisar firewall!)")
```

```
    else:
        print(f" Port {port} no específic")
else:
    print("IP externa - verificar regles de firewall")
```

Exemple pràctic: classificador de serveis

```
# classifica_servei.py
# Classifica un port en el servei corresponent

port = int(input("Introdueix el número de port: "))

if port == 21:
    servei = "FTP"
elif port == 22:
    servei = "SSH"
elif port == 25:
    servei = "SMTP"
elif port == 53:
    servei = "DNS"
elif port == 80:
    servei = "HTTP"
elif port == 443:
    servei = "HTTPS"
elif port == 3306:
    servei = "MySQL"
elif port == 5432:
    servei = "PostgreSQL"
elif 1 <= port <= 1023:
    servei = "Port reservat (sense nom específic)"
elif 1024 <= port <= 49151:
    servei = "Port registrat"
else:
    servei = "Port dinàmic/privat"

print(f"Port {port}: {servei}")
```

La instrucció match (Python 3.10+)

A partir de Python 3.10, l'alternativa moderna a seqüències llargues d'if/elif és la instrucció match. És especialment clara quan es compara un valor contra múltiples patrons concrets:

```
# Equivalent al classificador anterior amb match
port = int(input("Introdueix el número de port: "))

match port:
    case 21:
```

```

        servei = "FTP"
    case 22:
        servei = "SSH"
    case 25:
        servei = "SMTP"
    case 53:
        servei = "DNS"
    case 80:
        servei = "HTTP"
    case 443:
        servei = "HTTPS"
    case 3306:
        servei = "MySQL"
    case 5432:
        servei = "PostgreSQL"
    case _:
        servei = "Port no identificat"

print(f"Port {port}: {servei}")

```

match també permet combinar patrons amb `|` i fer *guards* amb `if`:

```

estat_http = 404

match estat_http:
    case 200 | 201 | 204:
        print("Èxit")
    case 301 | 302:
        print("Redirecció")
    case 400:
        print("Petició incorrecta")
    case 401 | 403:
        print("Accés denegat")
    case 404:
        print("Recurs no trobat")
    case code if 500 <= code <= 599:
        print(f"Error del servidor ({code})")
    case _:
        print("Codi desconegut")

```

NOTA

match no és un substitut de `if/elif` en tots els casos; és ideal quan es comparen valors discrets contra patrons. Per a rangs i condicions complexes, `if/elif` pot ser més clar.

3. Estructures repetitives: bucles

3.1. Estructura repetitiva while

El bucle while s'executa **mentre** una condició sigui certa.

```
while condicio:
    # bloc que es repeteix
```

```
# Compte enrere per reiniciar servei
comptador = 5
while comptador > 0:
    print(f"Reiniciant en {comptador}...")
    comptador -= 1
print("Reiniciant ara!")
```

Bucle amb entrada d'usuari

```
# Demanar contrasenya fins que sigui correcta
contrasenya_correcta = "linux2024"
intent = ""

while intent != contrasenya_correcta:
    intent = input("Contrasenya: ")
    if intent != contrasenya_correcta:
        print("Contrasenya incorrecta. Torna a intentar-ho.")

print("Accés concedit.")
```

3.2. Ús avançat de while

break, continue i else en bucles

```
# break: surt del bucle immediatament
intents = 0
max_intents = 3

while True:
    usuari = input("Usuari: ")
    contrasenya = input("Contrasenya: ")
    intents += 1

    if usuari == "admin" and contrasenya == "secret":
        print("Autenticació correcta")
        break

    if intents >= max_intents:
        print("Massa intents fallits. Compte bloquejat.")
        break

    print(f"Error. Intents restants: {max_intents - intents}")
```

```
# continue: salta a la propera iteració
i = 0
while i < 10:
    i += 1
    if i % 2 == 0:
        continue          # salta els parells
    print(i)              # imprimeix 1, 3, 5, 7, 9
```

La clàusula else en bucles

Els bucles for i while poden tenir una clàusula else que s'executa **només si el bucle no ha estat interromput per un break**. És molt útil per a cerques:

```
# Cercar un servei crític; si no es troba, avisar
serveis_actius = ["nginx", "sshd", "cron"]
servei_buscat = "mysqld"

for servei in serveis_actius:
    if servei == servei_buscat:
        print(f"Servei {servei_buscat} trobat i actiu.")
        break
else:
    # El bucle ha acabat sense trobar el servei (sense break)
    print(f"AVÍS: El servei {servei_buscat} NO està actiu!")
```

La instrucció pass

pass és una instrucció que no fa res. S'utilitza quan sintàcticament cal un bloc, però no volem executar cap acció (per exemple, funcions pendents d'implementar):

```
def monitoritza_xarxa():
    pass # TODO: implementar

# 0 en condicionals on de moment no cal acció
estat = "ok"
if estat == "ok":
    pass
elif estat == "error":
    print("Error detectat!")
```

Redirecció de fitxers per stdin

Des del terminal, podem fer que un script llegeixi d'un fitxer en comptes del teclat:

```
python3 processa_ips.py < llista_ips.txt
```

```
# processa_ips.py - llegeix IPs fins que acaba l'entrada
import sys

for linia in sys.stdin:
    ip = linia.strip()
    if ip:
        print(f"Processant: {ip}")
```

3.3. Estructura repetitiva for in

El bucle for recorre els elements d'una seqüència (llista, rang, cadena...).

```
servidors = ["web01", "web02", "db01", "mail01"]

for servidor in servidors:
    print(f"Comprovant {servidor}...")
```

range()

```
# range(fi)          -> 0, 1, ..., fi-1
# range(inici, fi)    -> inici, ..., fi-1
# range(inici, fi, pas) -> amb salt

for i in range(5):
    print(i)          # 0 1 2 3 4

for i in range(1, 6):
    print(i)          # 1 2 3 4 5

for i in range(0, 11, 2):
    print(i)          # 0 2 4 6 8 10
```

enumerate() --- índex i valor alhora

La documentació oficial recomana usar `enumerate()` en lloc de `range(len(...))` quan necessitem l'índex i el valor:

```
servidors = ["web01", "web02", "db01", "mail01"]

# Forma antiga (no recomanada)
for i in range(len(servidors)):
    print(i, servidors[i])

# Forma recomanada amb enumerate()
for i, servidor in enumerate(servidors):
    print(f" [{i}] {servidor}")

# Inici de l'índex personalitzat
for i, servidor in enumerate(servidors, start=1):
    print(f" {i}. {servidor}")
```

Exemple: escaneig de ports simulat

```
# escaneig_ports.py
# Simula un escaneig de ports comuns

ports_comuns = [21, 22, 23, 25, 53, 80, 110, 143, 443, 3306, 5432]

print("Ports a analitzar:")
for port in ports_comuns:
    if port in [22, 80, 443]:
        estat = "OBERT (comú)"
    elif port == 23:
        estat = "ATENCIO: Telnet (insegur)"
    else:
```

```
estat = "tancat"
print(f" Port {port:5d}: {estat}")
```

3.4. Estructura repetitiva i programació estructurada

Bucles imbricats

```
# Taula de xarxes possibles per a /24
octets = [192, 172, 10]

for primer in octets:
    for segon in range(168, 171):
        print(f" {primer}.{segon}.0.0/16")
```

Exemple: generador d'adreces IP

```
# Genera les primeres 5 IPs d'una xarxa /24
xarxa = "192.168.1"

print(f"Hosts a la xarxa {xarxa}.0/24:")
for host in range(1, 6):
    print(f" {xarxa}.{host}")
```

4. Tipus de dades compostes

4.1. Cadenes de text (str)

Les cadenes (*strings*) a Python són seqüències de caràcters immutables.

```
hostname = "servidor-web-01"
print(len(hostname))      # 15 (longitud)
print(hostname[0])        # "s" (primer caràcter)
print(hostname[-1])       # "1" (últim caràcter)
print(hostname[0:8])       # "servidor" (slice)
```

Mètodes de cadenes útils per a sysadmin

```
log = "  ERROR: Disc ple al servidor  "

print(log.strip())          # elimina espais als extrems
print(log.lower())          # tot en minúscules
print(log.upper())          # tot en majúscules
print(log.replace("ERROR", "CRÍTIC"))
print(log.startswith(" ERROR")) # True
print("Disc" in log)         # True
print(log.split(":"))        # [' ERROR', ' Disc ple al servidor  ']
```

4.2. Ús avançat de cadenes

Mètodes addicionals

```
ip = "192.168.1.100"
parts = ip.split(".")
print(parts)          # ['192', '168', '1', '100']
print(parts[0])        # '192'

# Reconstruir
nova_ip = ".".join(["10", "0", "0", "1"])
print(nova_ip)         # 10.0.0.1

# Comprovar contingut
port_str = "8080"
print(port_str.isdigit()) # True
print("abc".isdigit())    # False
```

Formatació avançada

```
# Alinear columnes en una taula
print(f"{'Servei':<15} {'Port':>6} {'Estat'}")
print("-" * 30)
serveis = [("SSH", 22, "actiu"), ("HTTP", 80, "actiu"), ("FTP", 21,
    ↪ "inactiu")]
for nom, port, estat in serveis:
    print(f"{nom:<15} {port:>6} {estat}")
```

Sortida:

Servei	Port	Estat
SSH	22	actiu
HTTP	80	actiu
FTP	21	inactiu

4.3. Llistes

Una **llista** és una col·lecció ordenada i modificable d'elements.

```
servidors = ["web01", "web02", "db01"]

# Accés per índex
print(servidors[0])      # "web01"
print(servidors[-1])     # "db01"

# Modificar
servidors[1] = "web02-nou"

# Afegir
servidors.append("mail01")
servidors.insert(1, "web02-backup")

# Eliminar
servidors.remove("db01")
eliminat = servidors.pop() # elimina i retorna l'últim

# Longitud
print(len(servidors))

# Comprovar existència
if "web01" in servidors:
    print("web01 és a la llista")
```

Operacions comunes

```
ports_oberts = [80, 443, 22, 8080, 3306]

# Ordenar
ports_oberts.sort()
print(ports_oberts)      # [22, 80, 443, 3306, 8080]

# Ordenar en sentit invers
ports_oberts.sort(reverse=True)

# Mínims i màxims
print(min(ports_oberts))
print(max(ports_oberts))

# Copiar una llista (important! NO usar: copia = ports_oberts)
copia = ports_oberts.copy()

# Mètodes addicionals (docs.python.org/3)
ports = [80, 443, 80, 22, 80]
print(ports.count(80))    # 3 --- comptar aparicions
print(ports.index(443))   # 1 --- posició de la primera ocurrència
```

```
altres = [8080, 8443]
ports.extend(altres)          # afegir tots els elements d'una altra
    ↪ llista
ports.clear()                 # buidar la llista
```

List comprehensions

Les *list comprehensions* són la forma **idiomàtica i recomanada** per la documentació oficial per crear llistes de forma concisa:

```
# Forma tradicional
ports_alts = []
for p in range(1024, 1030):
    ports_alts.append(p)

# List comprehension equivalent (recomanada)
ports_alts = [p for p in range(1024, 1030)]
print(ports_alts)  # [1024, 1025, 1026, 1027, 1028, 1029]

# Amb condició (filtrar)
ports_privilegiats = [p for p in range(1, 1024) if p in [22, 25, 53,
    ↪ 80, 443]]
print(ports_privilegiats)  # [22, 25, 53, 80, 443]

# Transformació: generar IPs d'una subxarxa
xarxa = "192.168.1"
hosts = [f"{xarxa}.{i}" for i in range(1, 6)]
print(hosts)
# ['192.168.1.1', '192.168.1.2', ..., '192.168.1.5']

# Filtrar servidors actius
servidors = [("web01", True), ("db01", False), ("mail01", True)]
actius = [nom for nom, estat in servidors if estat]
print(actius)  # ['web01', 'mail01']
```

4.4. Llistes i cadenes. Llistes multidimensionals. Tuples i Conjunts

Convertir entre llistes i cadenes

```
# Cadena → llista
ruta = "/etc/nginx/sites-enabled"
parts = ruta.split("/")
print(parts)    # ['', 'etc', 'nginx', 'sites-enabled']

# Llista → cadena
paraules = ["chmod", "755", "/var/www"]
ordre = " ".join(paraules)
print(ordre)    # chmod 755 /var/www
```

Llistes multidimensionals (matrius)

```
# Taula de servidors: [nom, ip, port]
xarxa = [
    ["web01", "192.168.1.10", 80],
    ["web02", "192.168.1.11", 80],
    ["db01", "192.168.1.20", 3306],
]

# Accés
print(xarxa[0])          # ['web01', '192.168.1.10', 80]
print(xarxa[0][1])       # '192.168.1.10'

# Recórrer
for servidor in xarxa:
    nom, ip, port = servidor
    print(f" {nom}: {ip}:{port}")
```

Tuples

Una **tupla** és com una llista, però **immutable**: un cop creada no es pot modificar. S'usa per a dades fixes o com a claus de diccionari:

```
# Crear tuples
connexio = ("192.168.1.1", 22, "SSH")
ip, port, protocol = connexio    # desempaquetatge (unpacking)
print(ip)                        # 192.168.1.1
print(port)                      # 22

# Tupla d'un sol element (cal la coma final!)
singleton = ("web01",)

# Les tuples es poden usar com a claus de diccionari (les llistes no)
regles_fw = {
    ("192.168.1.0", 80, "TCP"): "ALLOW",
```

```

    ("0.0.0.0", 23, "TCP"):      "DENY",
}
print(regles_fw[("0.0.0.0", 23, "TCP")])    # DENY

```

Conjunts (set)

Un **conjunt** és una col·lecció **sense duplicats** i **sense ordre**. Útil per eliminar repeticions o fer operacions matemàtiques de conjunts:

```

# Eliminar IPs duplicades d'un log
ips_log = ["10.0.0.1", "10.0.0.2", "10.0.0.1", "10.0.0.3", "10.0.0.2"]
ips_uniques = set(ips_log)
print(ips_uniques)    # {'10.0.0.1', '10.0.0.2', '10.0.0.3'}

# Operacions de conjunt
xarxa_a = {"web01", "web02", "db01"}
xarxa_b = {"db01", "mail01", "backup01"}

print(xarxa_a & xarxa_b)    # intersecció: {'db01'}
print(xarxa_a | xarxa_b)    # unió: tots els servidors
print(xarxa_a - xarxa_b)    # diferència: {'web01', 'web02'}

# Comprovar pertinença (molt ràpid)
ports_oberts = {22, 80, 443}
if 22 in ports_oberts:
    print("SSH accessible")

```

5. Coneixements bàsics de funcions

5.1. Introducció a les funcions

Una **funció** és un bloc de codi reutilitzable que duu a terme una tasca concreta. Permet:

- Evitar la repetició de codi (*DRY: Don't Repeat Yourself*)
- Organitzar el programa en parts lògiques
- Facilitar el manteniment i les proves

```

def saluda():
    print("Hola, administrador!")

# Cridar la funció
saluda()
saluda()

```

5.2. Primers programes amb funcions

Funcions amb paràmetres

```
def mostra_servidor(nom, ip):
    print(f"Servidor: {nom} ({ip})")

mostra_servidor("web01", "10.0.0.1")
mostra_servidor("db01", "10.0.0.2")
```

Funcions que retornen valors

```
def bytes_a_megabytes(bytes):
    return bytes / (1024 * 1024)

mida = bytes_a_megabytes(10485760)
print(f"Mida: {mida:.2f} MB")    # Mida: 10.00 MB
```

Paràmetres per defecte

```
def connecta(ip, port=22, protocol="SSH"):
    print(f"Connectant a {ip}:{port} via {protocol}")

connecta("192.168.1.1")           # usa defaults
connecta("192.168.1.1", 80, "HTTP") # tot explícit
connecta("192.168.1.1", port=443)  # paràmetre nombrat
```

Anotacions de tipus (*type hints*, Python 3.5+)

La documentació oficial recomana anotar els tipus dels paràmetres i el valor de retorn. No imposen restriccions en temps d'execució, però milloren la llegibilitat i permeten que eines com mypy detectin errors:

```
def bytes_a_megabytes(mida: int) -> float:
    """Converteix bytes a megabytes."""
    return mida / (1024 * 1024)

def valida_ip(ip: str) -> bool:
    """Retorna True si la cadena és una IP vàlida."""
    parts = ip.split(".")
    if len(parts) != 4:
        return False
    return all(p.isdigit() and 0 <= int(p) <= 255 for p in parts)

def connecta(ip: str, port: int = 22, protocol: str = "SSH") -> None:
    print(f"Connectant a {ip}:{port} via {protocol}")
```

f-strings avançades (Python 3.8+)

L'especificador `%` permet depurar mostrant el nom de la variable i el seu valor:

```
ip = "192.168.1.100"
port = 8080
actiu = True

# Depuració ràpida amb =
print(f"{ip=%}")      # ip='192.168.1.100'
print(f"{port=%}")    # port=8080
print(f"{actiu=%}")   # actiu=True

# Conversió de tipus amb !r i !s
nom = "web 01"
print(f"{nom!r}")      # 'web 01' (amb cometes, útil per logs)
print(f"{nom!s}")      # web 01 (equivalent a str())
```

5.3. Conceptes avançats de la programació modular

Àmbit de variables (scope)

```
servidor_global = "web01"  # variable global

def mostra():
    servidor_local = "db01"  # variable local (només visible aquí)
    print(servidor_global)   # pot llegir globals
    print(servidor_local)

mostra()
# print(servidor_local)  # ERROR: no existeix fora de la funció
```

Funcions que criden altres funcions

```
def valida_port(port):
    return 1 <= port <= 65535

def valida_ip(ip):
    parts = ip.split(".")
    if len(parts) != 4:
        return False
    for part in parts:
        if not part.isdigit():
            return False
        if not (0 <= int(part) <= 255):
            return False
    return True
```

```
def connecta_servei(ip, port):
    if not valida_ip(ip):
        print("IP invàlida")
        return
    if not valida_port(port):
        print("Port invàlid")
        return
    print(f"Connectant a {ip}:{port}...")

connecta_servei("192.168.1.1", 22)
connecta_servei("999.0.0.1", 22)
```

6. Mòduls i llistes de funcions

6.1. Introducció als mòduls

Un **mòdul** és un fitxer .py que conté funcions, variables i classes que podem reutilitzar en altres scripts.

Importar mòduls de la biblioteca estàndard

```
import os
import sys
import datetime

# Ús
print(os.getcwd())          # directori actual
print(sys.platform)         # sistema operatiu
print(datetime.date.today()) # data d'avui
```

Mòduls útils per a sysadmin

```
import os
import shutil
import subprocess
import socket

# Informació del sistema
print(f"Sistema: {os.name}")
print(f"CPU: {os.cpu_count()} nuclis")

# Nom de l'equip
print(f"Hostname: {socket.gethostname()}")

# Espai en disc
```

```
total, usat, lliure = shutil.disk_usage("/")
print(f"Disc - Total: {total//2**30} GB, Lliure: {lliure//2**30} GB")
```

Importació selectiva

```
from os.path import exists, getsize, join
from datetime import datetime

if exists("/etc/hosts"):
    mida = getsize("/etc/hosts")
    print(f"/etc/hosts: {mida} bytes")

ara = datetime.now()
print(f>Data i hora: {ara.strftime('%d/%m/%Y %H:%M:%S')}}")
```

6.2. Crear el nostre propi mòdul

Creem el fitxer `utils_xarxa.py`:

```
# utils_xarxa.py - Utilitats per a administradors de xarxa

def valida_ip(ip):
    """Comprova si una cadena és una adreça IP vàlida."""
    parts = ip.split(".")
    if len(parts) != 4:
        return False
    for part in parts:
        if not part.isdigit() or not (0 <= int(part) <= 255):
            return False
    return True

def ip_privada(ip):
    """Retorna True si la IP pertany a un rang privat."""
    if ip.startswith("10."):
        return True
    if ip.startswith("192.168."):
        return True
    if ip.startswith("172."):
        segon = int(ip.split(".")[1])
        return 16 <= segon <= 31
    return False

def port_conegut(port):
    """Retorna el nom del servei si és un port conegut."""
    serveis = {
        21: "FTP", 22: "SSH", 23: "Telnet",
        25: "SMTP", 53: "DNS", 80: "HTTP",
        110: "POP3", 143: "IMAP", 443: "HTTPS",
```

```

        3306: "MySQL", 5432: "PostgreSQL"
    }
    return serveis.get(port, "Desconegut")

```

Usar el mòdul des d'un altre script:

```

# main.py
import utils_xarxa

ip = input("IP: ")
if utils_xarxa.valida_ip(ip):
    if utils_xarxa.ip_privada(ip):
        print("IP privada (xarxa local)")
    else:
        print("IP pública")
else:
    print("Format d'IP invàlid")

```

6.3. Dunders i conceptes avançats

`__name__` i el bloc principal

```

# utils_xarxa.py (ampliat)

def valida_ip(ip):
    # ... codi ...
    pass

# Aquest bloc NOMÉS s'executa quan el fitxer s'executa directament,
# NO quan s'importa com a mòdul
if __name__ == "__main__":
    # Proves del mòdul
    print(valida_ip("192.168.1.1"))    # True
    print(valida_ip("999.0.0.1"))     # False

```

`__doc__` (docstrings)

```

def connecta(ip, port=22):
    """
    Connecta a un servidor per SSH.

    Paràmetres:
        ip (str): Adreça IP del servidor
        port (int): Port SSH (per defecte 22)

    Retorna:
        bool: True si la connexió ha estat exitosa
    """

```

```
"""
pass

print(connecta.__doc__)    # mostra la documentació
help(connecta)            # documentació formatada
```

7. Ús de funcions per la gestió de dades compostes

7.1. Llistes i algorismes d'ordenació

Ordenació amb `sort()` i `sorted()`

```
ports = [443, 80, 22, 3306, 53, 21]

# sort() modifica la llista original
ports.sort()
print(ports)    # [21, 22, 53, 80, 443, 3306]

# sorted() retorna una nova llista sense modificar l'original
nous_ports = sorted(ports, reverse=True)
print(nous_ports)
```

Ordenació per criteri personalitzat (`key=`)

```
servidors = [
    ("web01", 8, "actiu"),
    ("db01", 2, "inactiu"),
    ("mail01", 5, "actiu"),
]

# Ordenar per nombre de connexions (índex 1)
servidors.sort(key=lambda s: s[1], reverse=True)
for nom, conn, estat in servidors:
    print(f" {nom}: {conn} connexions ({estat})")
```

Algorisme de cerca lineal

```
def cerca_servidor(llista, nom_buscat):
    """Cerca un servidor per nom i retorna el seu índex o -1."""
    for i, servidor in enumerate(llista):
        if servidor == nom_buscat:
            return i
    return -1
```

```

servidors = ["web01", "web02", "db01", "mail01"]
pos = cerca_servidor(servidors, "db01")
if pos != -1:
    print(f"Servidor trobat a la posició {pos}")

```

7.2. Llistes de llistes

```

# Matriu de connectivitat entre servidors
# 1 = connexió permesa, 0 = bloquejada
firewall = [
    # web01 web02 db01 mail01
    [0, 1, 1, 0], # web01
    [1, 0, 1, 0], # web02
    [1, 1, 0, 0], # db01
    [0, 0, 0, 0], # mail01
]

servidors = ["web01", "web02", "db01", "mail01"]

def pot_connectar(origen, desti):
    i = servidors.index(origen)
    j = servidors.index(desti)
    return firewall[i][j] == 1

print(pot_connectar("web01", "db01")) # True
print(pot_connectar("mail01", "web01")) # False

```

7.3. Diccionaris

Un **diccionari** és una col·lecció de parells clau-valor. És ideal per representar registres estructurats.

```

servidor = {
    "nom": "web01",
    "ip": "192.168.1.10",
    "port": 80,
    "actiu": True,
    "serveis": ["nginx", "php-fpm"]
}

# Accés
print(servidor["nom"])
print(servidor["ip"])

# Modificar
servidor["port"] = 8080

# Afegir clau nova

```

```

servidor["so"] = "Debian 13"

# Eliminar clau
del servidor["port"]

# Comprovar existència
if "ip" in servidor:
    print(f"IP: {servidor['ip']}")

# Valor per defecte si no existeix
port = servidor.get("port", 80)

```

Mètodes principals

```

# Claus, valors i parells
print(servidor.keys())
print(servidor.values())
print(servidor.items())

# Recórrer el diccionari
for clau, valor in servidor.items():
    print(f" {clau}: {valor}")

```

7.4. Ús avançat de diccionaris

Diccionari de diccionaris (inventari de xarxa)

```

inventari = {
    "web01": {
        "ip": "192.168.1.10",
        "so": "Ubuntu 26.04",
        "ram_gb": 8,
        "serveis": ["nginx", "php8.4"]
    },
    "db01": {
        "ip": "192.168.1.20",
        "so": "Debian 13",
        "ram_gb": 16,
        "serveis": ["mariadb"]
    }
}

# Accés a dades niades
print(inventari["web01"]["ip"])
print(inventari["db01"]["serveis"])

# Afegir un nou servidor
inventari["mail01"] = {

```

```

        "ip": "192.168.1.30",
        "so": "Debian 13",
        "ram_gb": 4,
        "serveis": ["postfix", "dovecot"]
    }

# Mostrar resum
print("\nInventari de servidors:")
print(f"{'Nom':<10} {'IP':<16} {'RAM':>6} {'SO'}")
print("-" * 50)
for nom, info in inventari.items():
    print(f"{'nom':<10} {'info['ip']':<16} {'info['ram_gb']':>4} GB
        ↪ {'info['so']}")

```

Comptar elements amb diccionaris

```

# Comptar serveis únics a la xarxa
serveis_count = {}

for servidor, info in inventari.items():
    for servei in info["serveis"]:
        serveis_count[servei] = serveis_count.get(servei, 0) + 1

print("\nServeis instal·lats:")
for servei, count in sorted(serveis_count.items()):
    print(f" {servei}: {count} servidor(s)")

```

8. Ús de fitxers a Python per l'administrador de sistemes

8.1. Fluxos de fitxers, lectura i escriptura

Treballar amb fitxers és fonamental per a un administrador de sistemes: analitzar logs, llegir configuracions, generar informes...

Obrir un fitxer: `open()`

```
# Modes d'obertura
# 'r' → lectura (per defecte)
# 'w' → escriptura (sobreescriu si existeix)
# 'a' → afegir al final (append)
# 'x' → crear (error si ja existeix)
# 'r+' → lectura i escriptura

fitxer = open("fitxer.txt", "r")
# ... treballar amb el fitxer ...
fitxer.close()
```

Forma recomanada: `with`

El bloc `with` tanca el fitxer automàticament, fins i tot si hi ha errors:

```
with open("fitxer.txt", "r") as f:
    contingut = f.read()
    print(contingut)
# El fitxer es tanca automàticament en sortir del bloc with
```

Llegir fitxers

```
# Llegir tot el contingut
with open("/etc/hostname", "r") as f:
    nom = f.read().strip()
    print(f"Hostname: {nom}")

# Llegir línia a línia (eficient per fitxers grans)
with open("/etc/hosts", "r") as f:
    for linia in f:
        linia = linia.strip()
        if linia and not linia.startswith("#"):
            print(linia)

# Llegir totes les línies en una llista
with open("/etc/hosts", "r") as f:
    linies = f.readlines()
```

```
print(f"Total de línies: {len(linies)}")
```

Escriure fitxers

```
# Escriure (sobreescriu si existeix)
with open("informe.txt", "w") as f:
    f.write("Informe d'inventari\n")
    f.write("=" * 40 + "\n")
    f.write(f>Data: {datetime.date.today()}\n")

# Afegir al final (append)
with open("log.txt", "a") as f:
    f.write(f"[INFO] Servei iniciat correctament\n")
```

Exemple complet: analitzador de logs

```
# analitza_log.py
# Analitza un fitxer de log i mostra estadístiques

def analitza_log(ruta_fitxer):
    """Llegeix un fitxer de log i compta errors per tipus."""
    estadistiques = {
        "ERROR": 0,
        "WARNING": 0,
        "INFO": 0,
        "ALTRES": 0
    }
    total_linies = 0

    try:
        with open(ruta_fitxer, "r") as f:
            for linia in f:
                total_linies += 1
                linia = linia.strip()

                if "ERROR" in linia:
                    estadistiques["ERROR"] += 1
                elif "WARNING" in linia or "WARN" in linia:
                    estadistiques["WARNING"] += 1
                elif "INFO" in linia:
                    estadistiques["INFO"] += 1
                else:
                    estadistiques["ALTRES"] += 1

    except FileNotFoundError:
        print(f>Error: No es troba el fitxer '{ruta_fitxer}'")
        return
    except PermissionError:
```

```

        print(f"Error: Sense permisos per llegir '{ruta_fitxer}'")
        return

    print(f"\n=== Anàlisi de {ruta_fitxer} ===")
    print(f"Total de línies: {total_linies}")
    print()
    for tipus, count in estadistiques.items():
        pct = (count / total_linies * 100) if total_linies > 0 else 0
        print(f" {tipus:<10}: {count:>5} ({pct:.1f}%)" )

if __name__ == "__main__":
    import sys
    if len(sys.argv) < 2:
        print("Ús: python3 analitza_log.py <fitxer_log>")
        sys.exit(1)
    analitza_log(sys.argv[1])

```

Gestió d'excepcions en fitxers

```

import os

def llegeix_config(ruta):
    """Llegeix un fitxer de configuració clau=valor."""
    config = {}

    if not os.path.exists(ruta):
        raise FileNotFoundError(f"El fitxer de config no existeix:
        ↪ {ruta}")

    with open(ruta, "r") as f:
        for num_linia, linia in enumerate(f, 1):
            linia = linia.strip()
            if not linia or linia.startswith("#"):
                continue # saltar línies buides i comentaris

            if "=" not in linia:
                print(f" Advertència: línia {num_linia} ignorada
                ↪ (format incorrecte)")
                continue

            clau, valor = linia.split("=", 1)
            config[clau.strip()] = valor.strip()

    return config

# Exemple d'ús
try:
    config = llegeix_config("/etc/myapp.conf")

```

```
print(config)
except FileNotFoundError as e:
    print(f"Error: {e}")
```

Escriure informes amb print() redirigit a fitxer

```
import datetime

def genera_informe(inventari, ruta_sortida):
    """Genera un informe de text de l'inventari de servidors."""
    with open(ruta_sortida, "w") as f:
        def escriu(text=""):
            print(text, file=f)

        escriu("INFORME D'INVENTARI DE SERVIDORS")
        escriu("=" * 40)
        escriu(f"Generat: {datetime.datetime.now().strftime('%d/%m/%Y %H:%M')}")
        escriu()

        for nom, info in inventari.items():
            escriu(f"Servidor: {nom}")
            escriu(f"  IP   : {info['ip']}")
            escriu(f"  SO   : {info['so']}")
            escriu(f"  RAM  : {info['ram_gb']} GB")
            escriu(f"  Serveis: {', '.join(info['serveis'])}")
            escriu()

    print(f"Informe guardat a: {ruta_sortida}")
```

Exemple final: script d'inventari complet

```
#!/usr/bin/env python3
# inventari.py - Gestió d'inventari de servidors amb fitxers

import os
import datetime

FITXER_INVENTARI = "inventari.csv"

def carrega_inventari():
    """Carrega l'inventari des d'un CSV. Retorna diccionari."""
    inventari = {}
    if not os.path.exists(FITXER_INVENTARI):
        return inventari

    with open(FITXER_INVENTARI, "r") as f:
        for linia in f:
```

```

        linia = linia.strip()
        if not linia:
            continue
        parts = linia.split(",")
        if len(parts) >= 3:
            nom, ip, so = parts[0], parts[1], parts[2]
            inventari[nom] = {"ip": ip, "so": so}

    return inventari

def desa_inventari(inventari):
    """Desa l'inventari en format CSV."""
    with open(FITXER_INVENTARI, "w") as f:
        for nom, info in inventari.items():
            f.write(f"{nom},{info['ip']},{info['so']}\n")
    print(f"Inventari desat ({len(inventari)} servidors).")

def afegeix_servidor(inventari):
    """Afegeix un nou servidor a l'inventari."""
    nom = input("Nom del servidor: ").strip()
    if nom in inventari:
        print("Ja existeix un servidor amb aquest nom.")
        return
    ip = input("Adreça IP: ").strip()
    so = input("Sistema operatiu: ").strip()
    inventari[nom] = {"ip": ip, "so": so}
    print(f"Servidor '{nom}' afegit.")

def mostra_inventari(inventari):
    """Mostra tots els servidors."""
    if not inventari:
        print("L'inventari és buit.")
        return
    print(f"\n{'NOM':<15} {'IP':<18} {'SO':<10}")
    print("-" * 50)
    for nom, info in sorted(inventari.items()):
        print(f"{nom:<15} {info['ip']:<18} {info['so']:<10}")
    print(f"\nTotal: {len(inventari)} servidors")

def main():
    inventari = carrega_inventari()

    while True:
        print("\n=== Gestió d'Inventari ===")
        print("1. Mostrar inventari")
        print("2. Afegir servidor")
        print("3. Desar i sortir")

        opcio = input("\nOpció: ").strip()

        if opcio == "1":
            mostra_inventari(inventari)
        elif opcio == "2":
            afegeix_servidor(inventari)

```

```

        afageix_servidor(inventari)
    elif opcio == "3":
        desa_inventari(inventari)
        print("Fins aviat!")
        break
    else:
        print("Opció no vàlida.")

if __name__ == "__main__":
    main()

```

Guardar dades estructurades amb JSON

La documentació oficial recomana el mòdul `json` per desar i carregar dades estructurades (diccionaris, llistes...) de forma estàndard i interoperable:

```

import json
import os

FITXER_JSON = "inventari.json"

# Dades d'exemple
inventari = {
    "web01": {"ip": "192.168.1.10", "so": "Ubuntu 26.04", "serveis":
        ↪ ["nginx"]},
    "db01": {"ip": "192.168.1.20", "so": "Debian 13", "serveis":
        ↪ ["mariadb"]},
}

# Escriure JSON (desar)
with open(FITXER_JSON, "w", encoding="utf-8") as f:
    json.dump(inventari, f, indent=2, ensure_ascii=False)
print(f"Desat a {FITXER_JSON}")

# Llegir JSON (carregar)
if os.path.exists(FITXER_JSON):
    with open(FITXER_JSON, "r", encoding="utf-8") as f:
        dades = json.load(f)
    print(dades["web01"]["ip"]) # 192.168.1.10

```

El fitxer `inventari.json` resultant té aquest aspecte, llegible per qualsevol eina o llenguatge:

```

{
  "web01": {
    "ip": "192.168.1.10",
    "so": "Ubuntu 26.04",
    "serveis": ["nginx"]
  },
  "db01": {

```

```

    "ip": "192.168.1.20",
    "so": "Debian 13",
    "serveis": ["mariadb"]
}
}

```

NOTA

Avantatge de JSON vs. CSV: JSON preserva l'estructura (l·listes dins diccionaris) sense haver de fer parsing manual. Per a inventaris complexos és l'opció preferida.

Resum de conceptes clau

Concepte	Sintaxi	Exemple
Variable	nom = valor	port = 22
Entrada	input("text")	ip = input("IP: ")
Sortida	print(...)	print(f"Port: {port}")
Condicional	if / elif / else	if port == 22:
match (3.10+)	match val: case x:	match port: case 22:
Bucle mentre	while condicio:	while actiu:
Bucle per	for x in seq:	for s in servidors:
else en bucle	for/while ... else:	s'executa si no hi ha break
Funció	def nom(params):	def valida_ip(ip: str) -> bool:
List comp.	[x for x in seq if cond]	[p for p in ports if p < 1024]
Llista	[a, b, c]	[80, 443, 22]
Tupla	(a, b, c)	("web01", "10.0.0.1", 22)
Conjunt	{a, b, c}	{22, 80, 443}
Diccionari	{k: v}	{"ip": "10.0.0.1"}
Fitxer text	with open(...) as f:	with open("log", "r") as f:
Fitxer JSON	json.dump / json.load	json.dump(dades, f, indent=2)

Recursos addicionals

- [Documentació oficial de Python 3](#)
- [W3Schools Python Tutorial](#)
- [IOC - Programació Bàsica \(ASIX\)](#)
- [Real Python](#) -- tutorials pràctics en anglès

Índex de figures

1 Python logo 1

Versions d'aquest document

- [HTML](#)
- [PDF](#)
- [ODT](#)
- [MD](#)

[Domini Públic \(CC0\)](#)