Introducció a l'algorísmica

Algorísmica

Grau d'Enginyeria Informàtica

Mireia Ribera, ribera@ub.edu
Daniel Ortiz, daniel.ortiz@ub.edu

Introducció a l'algorísmica: Conceptes clau

1. Assignatura, funcionament i avaluació
2. Què és un algorisme? Entrada, sortida. Eficiència i correcció
3. Què és un error i quins tipus hi ha
4. Llenguatges de programació: sintaxi i semàntica

Assignatura, funcionament i avaluació

Teoria

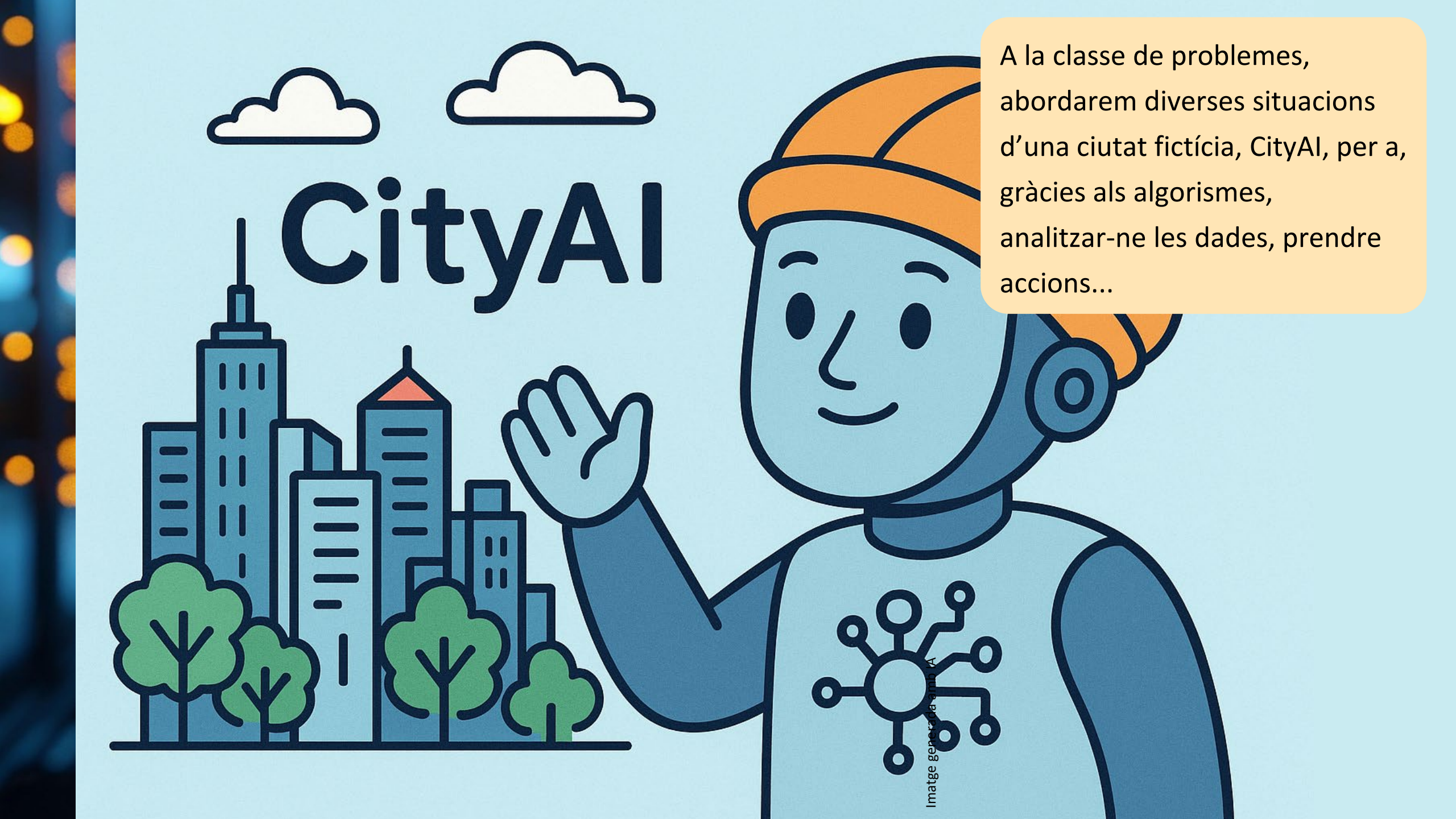
- Explicació dels conceptes de l'assignatura
- Demostracions i exemples
- Examen parcial i final
- 1.5h a la setmana

Problemes

- Resolució en grup
- Dubtes de Teoria
- Exemples
- 1.5h a la setmana

Laboratori

- Resolució individual
- Prova1: 20,23-oct-2025
- Prova2: 17,20-nov-2025
- Prova3: 15,18-des-2025
- Correcció portfoli
- 1.5h a la setmana



A la classe de problemes, abordarem diverses situacions d'una ciutat fictícia, CityAI, per a, gràcies als algorismes, analitzar-ne les dades, prendre accions...

Funcionament de les classes de teoria

Activitats a l'aula

- Sessions teòriques
- Apunts en PDF
- Demostracions en Jupyter Notebook
- Exàmens amb part teòrica i pràctica que cal >2.5 per aprovar.

Activitats fora de l'aula i recursos

- Activitats per realitzar abans de la propera sessió
- Recursos addicionals optatius
- Tutories sota demanda

Interacció

- Preguntes en el mòbil
- Les respostes són anònimes. No s'avalua ningú.

Funcionament de les classes de problemes i laboratori

Eines

- Treballarem a Windows
- Usarem Jupyter Notebook
- També podem usar PyCharm com a suport pel codi.

Enunciats

- Xuleta d'exàmens
- Enunciats i algunes solucions d'exemple
- Llibre "Problemes bàsics d'algorísmica.

Avaluació (laboratori)

- No funciona:0
- Dades públiques: 2.5
- Dades privades: 5
  - + 2 eficiencia
  - + 2 elegància
  - + 1 específic
- Repte: +0.5

Avaluació de l'assignatura: Avaluació contínua

- Nota laboratori(LP): 3 punts sobre la nota final. Cal treure>4 per superar l'assignatura
  - Prova1: 0.5 punts; Prova2: 0.75 punts; Prova3: 0.75 punts; Portafolis: 1 punt
  - Nota portafolis: numero problemes entregats a temps (0 a 5), correcció de dos problemes aleatoris (0 a 5)
- Nota teoria(NT): 7 punts sobre la nota final. Cal treure>4 per superar l'assignatura
  - Examen parcial (EP): 3 punts, 2 oportunitats 4-novembre-2025 i 20-gener-2025 (si s'ha suspès primera oportunitat)
  - Examen final (EF): 4 punts, 20-gener-2025
  - Si part pràctica o part teòrica (sobre 5) <=2.5, nota = min (part pràctica, part teòrica)
- Si no s'ha tret un 4 en alguna de les dues, la nota final és min(4,0; (LP + NT)).
- Als exàmens de teoria i laboratori no es pot usar IA

Avaluació de l'assignatura: Reavaluació i avaluació única

- Si la nota d'avaluació contínua >= 3.5, es pot fer un examen de reavaluació, un únic examen que avalua tota l'assignatura.
  - A l'examen si part pràctica o part teòrica <=2.5, nota = min (part pràctica, part teòrica)
- Si l'alumne ho sol·licita en el calendari establert pot acollir-se a l'avaluació única.

Notebook de suport

Calculeu les vostres notes amb el notebook: IntroTeoriaCalculNotes.ipynb

Llibres que han inspirat l'assignatura o que usarem

- M. Ribera, J. Vitrià, P. Laiz i P. Gilabert Problemes bàsics d'algorísmica Publicacions UB, 2021.
- T. H. Cormen et al. Introduction to algorithms, 4th ed. MIT Press, 2022.
- S.Dasgupta.Algorithms, McGrawHill, 2006.
- V. Levitin, Introduction to the Design and Analysis of Algorithms. 3rd ed. Addison-Wesley, 2014.
- S. Skiena. The Algorithm Design Manual. 3rd ed. Springer, 2020
- Louridas Algorithms (The MIT Press essential knowledge guides). MIT Press, 2020
- Python Programming for Beginners. AMZ Publishing, 2021
- Bhargava, Aditya Grokking Algorithms: an illustrated guide for programmers and other curious people. Manning, 2016.
- Spraul, V. Anton Think Like a programmer: an introduction to creative problem solving. No Starch Press, 2012

Què és un algorisme? Entrada, sortida. Eficiència i correcció

Un algorisme és qualsevol ***procediment computacional*** que

pren un (o una sèrie) de dades/valors com a ***entrada***

i genera alguna dada/valor (o sèrie de dades/valors) com a ***sortida***.

- Els algorismes són les idees/estratègies que hi ha darrera els programes per resoldre un determinat problema.
- Els algorismes són **independents del llenguatge** en que estan escrits. El mateix algorisme escrit en dos llenguatges diferents pot tenir una aparença superficial molt diferent.
- Els algorismes **sí que depenen de la representació de les dades**.
- Els algorismes interessants són els que resolen problemes generals.

Exemple: Multiplicació d'enters

Quan definim un problema (per exemple, la multiplicació de dos nombres) a resoldre mitjançant un algorisme, ens cal especificar tres elements:

- L'entrada: En el cas de la multiplicació l'entrada seran dos nombres enters, *a* i *b*.
- Quina sortida esperem: Un nombre *c* tal que $c = a * b$.
- Quins requeriments imposem: La solució ha de ser **correcta** i **eficient**.

Hi ha **diversos algorismes** que resolen aquest problema.

El que heu vist a l'escola és un d'ells, però hi ha un algorisme "producte rus" que també la resol.

Exemple: Multiplicació d'enters – Algorisme Rus

En aquest algorisme, a i b van canviant de valor fins arribar a la solució. En primer lloc a = a dividit per 2, si el mòdul de la divisió és 1, el resultat s'incrementa en b. b = b multiplicat per 2 .I així fins que el primer operand és 0. Per exemple amb 47 * 12, obtenim 564

a = 47	b = 12	resultat
47 Dividim //2 mentre a>0	b = 12 Multipliquem x2	
a = 47 // 2 = 23 ; mòdul 1	b = 12 * 2 = 24	12 Si a%2==1 sumem b
a = 12 // 2 = 11 ; mòdul 1	b = 24 * 2 = 48	36 sumem
a = 11 // 2 = 5 ; mòdul 1	b = 48 * 2 = 96	84 sumem
a = 5 // 2 = 2 ; mòdul 1	b = 96 * 2 = 192	180 sumem
a = 2 // 2 = 1 ; mòdul 0	b = 192 * 2 =384	180 no sumem
a = 1 // 2 = 0 ; mò0dul 1	b = 384 * 2 = 768	**564 sumem**

Algorisme Rus – Codificació en Python

```python
def producte_rus(a: int, b: int) -> int:
    """ Multiplica dos enters a i b utilitzant l'algorisme del producte rus. Retorna el resultat d’ a * b. """
    resultat = 0
    while a > 0:
        # Si a és senar, afegeix b al resultat
        if a % 2 == 1:
            resultat += b
        # Divideix a per 2 (divisió entera) i multiplica b per 2
        a //= 2
        b *= 2
    return resultat
```

Correcció

Un algorisme és **correcte** si podem demostrar (matemàticament) que retorna la sortida desitjada per qualsevol entrada legal.

Demostrar la correcció a vegades és fàcil, però sovint és difícil o impossible.

Per exemple, en el cas de l'algorisme rus, si considerem només positius i els representem en binari, tenim que $a = \sum_{i=0}^{k} \alpha_i 2^i \ \ amb\ \alpha_i \in \{0,1\}$ i llavors $a \cdot b = \sum_{i=0}^{k} \alpha_i \ \left(2^i \cdot b\right)$. L'algorisme recorre exactament els valors $2^i b$ duplicant b i suma al resultat aquells termes corresponents a $\alpha_i = 1$ quan a té el bit i en 1.

A l'exemple, amb a=47, la seva representació binària és 1011111

Exemple: Multiplicació d'enters – Algorisme Rus

En aquest algorisme, es divideix el primer operand per 2, si el mòdul és 1, se suma el segon operand al resultat. El segon operand es multiplica per 2. I així fins que el primer operand és 0. Per exemple amb 47 * 12, obtenim 564

a = 47 = 1011111	b	resultat	
47 Dividim //2	12 Multipliquem x2		0, 1
23	24	12 Si a%2==1 sumem b	1, 1
11	48	36	2, 1
5	96	84	3, 1
2	192	180	4, 1
1	384	180	5, 0
0 Repetim si a>0	768	**564**	6, 1

Eficiència

Un algorisme és eficient si es fa amb el mínim nombre de recursos de temps – cicles de rellotge -, i d'espai – memòria -.

Fer servir algorismes eficients és sempre convenient i a vegades és una necessitat!

Per exemple, en el cas de l'algorisme rus, les operacions de multiplicació per 2 o de divisió per 2 en binari són molt eficients, ja que només signifiquen moure els bits a l'esquerra o a la dreta una posició (veure exemple). Mirar si un nombre és senar només consisteix a mirar si el seu últim bit és un 1, i sumar, com veurem més endavant també és una operació eficient.

Exemple:

multiplicació binària 110 x 2 = 1100, divisió entera binària 110 // 2 = 10 = 11, 111 // 2 = 11

Algorismes i ordinadors

Un ordinador només fa dues coses:

- Calcular (combinar dades per obtenir altres dades)
- Emmagatzemar (llegir i escriure a una memòria).

Un ordinador de sobretaula actual, amb 8 nuclis i 4GHz, fa fins a 256.000.000.000 càlculs per segon. Un ordinador amb 32GB de RAM pot emmagatzemar fins a 256.000.000.000 bits a memòria. I amb un disc dur d'1 TB pot emmagatzemar 8.000.000.000.000 bits.

En aquest curs veurem algorismes basats en un ordinador de sobretaula convencional. No veurem algoritmes:

- Basats en càlcul paral·lel ni distribuït entre diversos ordinadors
- Basats en arquitectures no convencionals (quàntiques)

El problema de la motxilla

Definició del problema

La motxilla pot portar un pes màxim de 2.3kg

Disposem d'un seguit d'objectes que hi volem posar, cadascun amb un pes $wobj_i$ i amb un valor $vobj_i$. Exemple: la bufanda pesa 0.2kg (wobj) i té un valor de 30€ (vobj).

Volem triar aquells elements que maximitzin el valor de la motxilla un cop carregada.

Aquest és un problema genèric per a portafolis financers, càrregues de vaixells o avions, tall de materials o emplenat de contenidors.

Quina és l'*entrada*? Quina la *sortida esperada*? Com es defineix *correcte*? i *eficient*?

Solució 1

Anem posant els objectes tal i com ens arriben fins a emplenar la motxilla

És correcte? és *eficient*?

Solució 2

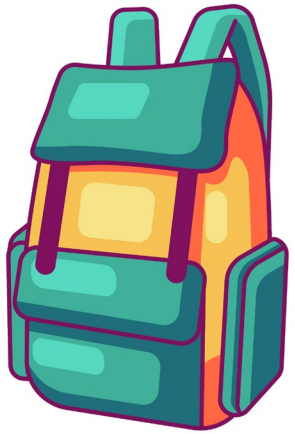
Considerem totes les combinacions possibles d'elements i agafem la combinació més valuosa.

És correcte? és *eficient*?

Considerem totes les combinacions possibles d'elements. *Eliminem les que pesen més del màxim establert*. Triem la combinació més valuosa.

És correcte? és *eficient*?

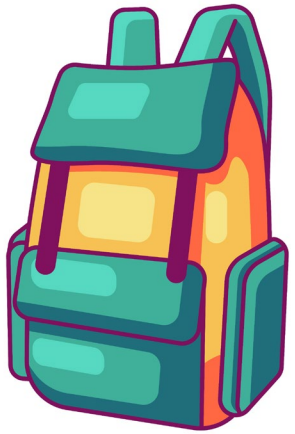
Càlcul



2.3kg

Botes 2kg 150€	Mòbil 0.4kg 300€	Bufanda 0.2Kg 30€	Crema 0.5Kg 50€	Pes	Valor
0	0	0	1	0.5kg	50
0	0	1	0	0.2kg	30
0	0	1	1	0.7kg	80
0	1	0	0	0.4kg	300
0	1	0	1	0.9kg	350
0	1	1	0	0.6kg	330
0	1	1	1	1.1kg	380
1	0	0	0	2kg	150
1	0	0	1	2.5kg	200
1	0	1	0	2.2kg	180
1	0	1	1	2.7kg	230
1	1	0	0	2.4kg	450
1	1	0	1	2.9kg	500
1	1	1	0	2.6kg	480
1	1	1	1	3.1kg	530

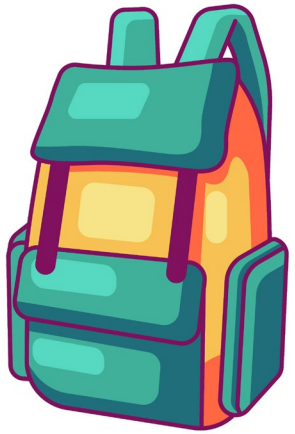
Càlcul



2.3kg

Botes 2kg 150€	Mòbil 0.4kg 300€	Bufanda 0.2Kg 30€	Crema 0.5Kg 50€	Pes	Valor
0	0	0	1	0.5kg	50
0	0	1	0	0.2kg	30
0	0	1	1	0.7kg	80
0	1	0	0	0.4kg	300
0	1	0	1	0.9kg	350
0	1	1	0	0.6kg	330
0	1	1	1	1.1kg	380
1	0	0	0	2kg	150
1	0	0	1	2.5kg	200
1	0	1	0	2.2kg	180
1	0	1	1	2.7kg	230
1	1	0	0	2.4kg	450
1	1	0	1	2.9kg	500
1	1	1	0	2.6kg	480
1	1	1	1	3.1kg	530

Càlcul



2.3kg

Botes 2kg 150€	Mòbil 0.4kg 300€	Bufanda 0.2Kg 30€	Crema 0.5Kg 50€	Pes	Valor
0	0	0	1	0.5kg	50
0	0	1	0	0.2kg	30
0	0	1	1	0.7kg	80
0	1	0	0	0.4kg	300
0	1	0	1	0.9kg	350
0	1	1	0	0.6kg	330
0	1	1	1	1.1kg	380
1	0	0	0	2kg	150
1	0	0	1	2.5kg	200
1	0	1	0	2.2kg	180
1	0	1	1	2.7kg	230
1	1	0	0	2.4kg	450
1	1	0	1	2.9kg	500
1	1	1	0	2.6kg	480
1	1	1	1	3.1kg	530

Com expressem els algorismes

Amb llenguatges de programació.

Un llenguatge de programació es defineix per unes primitives (símbols), una sintaxi (regles de combinació de símbols), una semàntica estàtica (combinacions de símbols amb significat) i una semàntica (el significat que nosaltres volem donar a l'algorisme).

Ex. A Python def és un símbol, la sintaxi de print és posar l'argument entre parèntesi, no podem fer "hola" + 4 (semàntica estàtica), i l'algorisme del producte rus resol la multiplicació.

Com escriure bé

Quan programem, pot ser que cometem errors, per això les eines de programació ens ajudaran a detectar errors.

Els errors inicials més comuns seran errors sintàctics, consistents en no escriure bé les instruccions, no posar els espais que toquen, no posar parèntesi... Aquests són els errors més fàcils de detectar i corregir pels entorns de programació.

A vegades també cometrem errors de semàntica estàtica, combinant variables o fent operacions que no són correctes. Alguns d'aquests errors també es poden detectar automàticament.

Errors

Si no hi ha errors sintàctics ni de semàntica estàtica, el programa s'executarà i farà alguna cosa, però no necessàriament el que volem, estarem cometent errors de semàntica. Si ens ajudem de la IA per programar, ens pot proposar algorismes amb errors semàntics també.

Els errors semàntics són molt difícils de detectar de manera automàtica:

La correctesa semàntica del codi és **responsabilitat del o de la programador/a.**

```
def suma(a:int, b:int)->int:
    """ Aquest programa no fa el què ha de fer!"""
    return a - b
```

Gràcies i recapitulació

MIREIA RIBERA | ACCESSIBILITAT DIGITAL
| EXPERIÈNCIA D'USUARI
| VISUALITZACIÓ DE DADES

UNIVERSITAT DE BARCELONA

Recapitulació

📄 Cal consultar el pla docent per conèixer com s'avalua l'assignatura

📚 Teoria, problemes i laboratori es complementen per afavorir l'aprenentatge

🧩 Els algorismes són solucions correctes i eficients a problemes generals

⚠️ Els errors semàntics són els més difícils de detectar i són responsabilitat del/de la programador/a.

🐍 Aquest curs programarem amb Python i amb Jupyter Notebooks

⏱️ Cal aprendre les estructures bàsiques del llenguatge

Python I: Conceptes clau

1. Instrucció, variable, literal i expressió. Referències a variables. Comentaris
2. Definir, mostrar o retornar resultats.
3. Funció i mètode. Paràmetres. Cridar o invocar
4. Assignacions. Iteracions. Condicionals
5. Ús de ChatGPT o IA generativa

Python

En aquest curs usarem Python amb Jupyter Notebooks (recordeu el que vàreu veure al Seminari d'Introducció a la Programació).

Quan escrivim un programa a Python hem de cuidar especialment dos elements:

- **Noms o Identificadors**: Els fem servir per anomenar funcions i variables.
  - Han de començar per lletra o _ No poden contenir espais
  - Distingeixen entre majúscules i minúscules
  - Hi ha noms reservats (and, for, def)
- **Expressions**: És el codi que calcula o produeix nous valors de les dades.
  - L'expressió més simple és el literal. Pot ser un valor o una variable.
  - Una expressió pot ser una combinació d'expressions més simples amb operadors.
  - Els operadors segueixen la precedència estàndard.

Python: variables

- Les variables són espais de memòria on es guarden dades.
- Cada variable té un identificador.
- Cada variable és d'un tipus determinat (enter, decimals, booleà...) i ocupa un espai determinat. Python és un llenguatge molt flexible amb els tipus però a Algorísmica tipem explícitament.
- Si definim una variable dins un bloc de codi només la podem usar dins aquell bloc de codi (àmbit de la variable).

Python: Instrucció, variables, literals i expressions

```
# Això és un comentari: el programa no ho executa
# Instrucció: assignem un valor a una variable
edat: int = 25 # 'edat' és la variable, int vol dir integer i és el tipus, 25 és un
literal
# Una altra instrucció: imprimir per pantalla
print("Hola món!") # "Hola món!" és un literal de text (string)
# Expressió: suma d'una variable i un literal
nova_edat: int = edat + 5 # 'edat + 5' és una expressió que dona com a resultat 30
# Els operadors són +, -, *, /, // (divisió entera), ** (potència), % (mòdul)
# Mostrar el resultat
print("D'aquí 5 anys tindràs:", nova_edat)
```

Per tal que els nostres programes siguin llegibles i clars, seguirem unes pautes d'escriptura estandarditzades a Python, les directrius PEP8. Vegeu el notebook EstilPogramació.ipynb del Seminari d'introducció a la programació.

Python: referències

A Python cada variable té associada una referència, un identificador, que d'alguna manera ens diu on es pot localitzar. Segons el tipus de variable aquestes referències tenen un funcionament diferent.

```
a:int = 4
b:int = 5
c:int = 6
print(id(a), id(b), id(c))
d:str = "banana"
e:str = "banana"
print(id(d),id(e))
f:list[int] = [1, 2, 3]
g:list[int] = [1, 2, 3]
print(id(f),id(g))
```

Vegeu el notebook Lab2-Referencies.ipynb de Laboratori

Python: clonatge de variables

Si dues variables apunten a la mateixa referència, les modificacions poden ser perilloses.

Per evitar efectes indesitjats, quan copiem una llista caldrà clonar-la

```
a:list[int] = [1, 2, 3]
b:list[int] = a
print(a,b,id(a),id(b))
b[2] = 4
print(a)

>> [1, 2, 3] [1, 2, 3] 1673421612160 1673421612160
>> [1, 2, 4]
```

```
a:list[int] = [1, 2, 3]
b:list[int] = a[:]  # b ja no apunta a "a", n'ha fet un "clon"
print(a,b,id(a),id(b))
b[2] = 4
print(a)

>> [1, 2, 3] [1, 2, 3] 1673421611072 1673420642368
>> [1, 2, 3]
```

Python: funcions, entrada i sortida

Recordem que un algorisme és "*aquell procediment computacional que pren un (o una sèrie) de dades/valors com a entrada i genera alguna dada/valor (o sèrie de dades/valors) com a sortida*."

Les **dades o els valors que donem** a un algorisme és el que anomenem paràmetres. Per cada paràmetre haurem d'indicar un nom de variable i el tipus.

Les **dades o valors de sortida** que ens genera l'algorisme els podem:

- Mostrar per pantalla (print) – els veurem
- Retornar (return) - vol dir que jo puc assignar a una variable el resultat de l'algorisme. Ex:

```
preu:int = calcul_IVA(100)
```

Python: crida de funcions

Quan Python rep la crida d'una funció, fa quatre coses:

- El programa que fa la crida se suspèn/congela en el punt de la crida.
- Els paràmetres de la funció passen a prendre els valors de la crida.
- S'executa el cos de la funció.
- Retorna el control al punt de programa posterior a la crida.

```
def sum(a:int, b:int) -> int:
    return a + b
def dif(a:int, b:int) -> int:
    return a - b
```

```
def sumdif(a:int, b:int) -> int:
    s:int = sum(a, b)
    d:int = dif(a, b)
    return s, d
```

```
a:int = 3
b:int = 3
print(sumdif(a, b))
```

Python: definir (def), cridar, mostrar resultats(print) , retornar resultats(return)

```python
def suma(a: int, b: int) -> int:  #només definim, no executem
    print("la suma és ", a+b)
    return a + b

c: int = suma(5,7)  # cridem la funció, c pren el valor del return

def dos_valors(a: int) -> tuple[int, int]:
      return a, a+2

d: int
e: int
d, e = dos_valors(4)  # cridem la funció, c, d prenen el valor del return
print(c,d,e)
```

Python: funció, mètode i acció

```python
# 1 FUNCIÓ: bloc de codi que
  retorna un resultat
def suma(a: int, b: int) -> int:
    return a + b
resultat: int = suma(3, 4) #Crida


# 2 MÈTODE: forma part d'un objecte
  o tipus. La sintaxi és diferent.
text: str = "hola món"
majuscules: str = text.upper()
print("Text en majúscules ",
  majuscules)
```

```python
# 3 FUNCIÓ SENES RETORN: encara que retorni
None és una funció

def saluda() -> None:
    print("Hola! Encantat de conèixer-te!")
    # Només mostra el resultat per pantalla

saluda() # Crida la funció però no retorna res


len(text)


len és una funció de Python que retorna la
llargada en caràcters de les cadenes de
caràcters. No la definim nosaltres.
```

Python: paràmetre, crida o invocació

```python
# 1 Definim una funció amb dos paràmetres
def suma(a: int, b: int) -> int:
    return a + b


# 2 Definim una funció sense paràmetres (funció que no necessita informació de fora)
def missatge() -> str:
    return "Hola!"
```

```python
resultat1:int = suma(3, 4)
# passem 3 i 4 directament
print("Cas 1:", resultat1) #

x: int = 10
y: int = 5
resultat2: int = suma(x, y)
# passem el contingut de x i y
print("Cas 2:", resultat2) #

text: str = missatge()
# no necessita informació extra
print("Cas 3:", text)
```

Ara et toca...

Escriu una funció que retorni la multiplicació de tres nombres enters

Escriu una altra funció que mostri per pantalla 5 vegades “hola” i retorni “molt de gust”

Usa el mètode .count de str per saber quantes “a” hi ha en una paraula

Python: variables

Recordem: les variables són espais de memòria per guardar-hi valors. Cada variable és d'un tipus i accepta uns determinats valors.

Aprenem noves coses: per donar valors a les variables podem fer

- assignacions directes al codi, amb literals, expressions o funcions,
- demanar-li dades a l'usuari (assignacions d'entrada).
  - Sempre "llegirem" una cadena i l'haurem de convertir al tipus que volem.
  - La conversió es pot fer amb eval o directa
- assignar simultàniament valors a més d'una variable.

Python: assignacions

```python
# --- ASSIGNACIONS AMB LITERALS ---

numero: int = 10        # enter

preu: float = 19.99     # nombre decimal

nom: str = "Maria"      # text (string)

es_major: bool = True   # booleà


# --- ASSIGNACIONS AMB EXPRESSIONS ---

suma: int = 5 + 3     # resultat = 8

producte: int = numero * 2  # resultat = 20

preu_final: float = preu * 1.21  # aplica IVA

nom_complet: str = nom + " López" # concatenació

gran: bool = numero > 5 # comparació
```

```python
# --- ASSIGNACIÓ AMB CRIDA A FUNCIÓ ---

longitud_nom: int = len(nom)
# len() retorna la longitud de la cadena
valor_abs: int = abs(-42)
# abs() retorna el valor absolut
maxim: int = max(10, 3, 25)
# màxim de tres valors

# --- ASSIGNACIÓ MÚLTIPLE ---
x: int           # tipus primer
y: int
x, y = 1, 2      # assigna 1 a x i 2 a y

a: str           # tipus primer
b: str
c: str
a, b, c = "A", "B", "C"
```

Python: assignacions d'entrada. Ús d'eval o conversió directa

```
# --- TEXT (string) ---
nom1: str = input("Introdueix el teu nom: ")
nom2: str = eval(input("Introdueix el teu nom entre cometes: "))

  # --- ENTER (int) amb conversió directa ---
edat1: int = int(input("Introdueix la teva edat: "))
edat2: int = eval(input("Introdueix la teva edat: "))

  # --- DECIMAL (float) amb conversió directa ---
altura1: float = float(input("Introdueix la teva altura en metres: "))
altura2: float = eval(input("Introdueix la teva altura en metres: "))

print(nom1,nom2,edat1,edat2,altura1,altura2)
```

Python: iteracions

Per no repetir una instrucció moltes vegades podem fer servir l'estructura de control de la iteració (bucles, loops, ...).

Hi ha dos tipus d'iteracions bàsics a Python:

- Iteracions for: recorren una llista de valors establerta
- Iteracions while: es repeteix el codi mentre es compleix una condició.

Vegeu els notebook Lab1-1RevEstructuresControl.ipynb, i Lab1-2EstructuresControl.ipynb

Python: iteracions for: range(start (0), stop (end), step (1))

```python
# Recorregut sobre una llista
Colors: list[str] = ["vermell",
  "blau", "verd"]
for color in colors:
    print(color)


# Recorregut sobre una paraula
for lletra in "Hola":
    print(lletra)
```

```python
# Amb range(): números del 0 al 4
for i in range(5):
    print(i)

# Amb range(): del 2 al 10 saltant
de 2 en 2
for i in range(2, 11, 2):
    print(i)
```

Python: iteracions while

```python
# Demanem un número fins que sigui positiu
num: int = int(input("Introdueix un número positiu: "))
while num <= 0:
    num = int(input("Introdueix un número positiu: "))
print("Has introduït:", num)
```

For: sabem quan s'aturarà. Millor per a recorreguts.

While: desconeixem quan s'aturarà, depèn dels valors en el moment d'execució. Millor per a cerques

Python: condicionals

Per executar codis alternatius segons el valor d'una condició podem fer servir l'estructura de control condicional.

Aquestes estructures tenen diversos elements:

- Sempre comencen per if: a if indiquem la condició inicial i posem el codi que s'executarà si es compleix
- Quasi sempre tenen un else: a else posem el codi que s'executarà si la condició NO es compleix.
- Poden tenir un o més elif(else + if): amb noves condicions per proveir de més alternatives.

Python: condicionals (estructures de control)

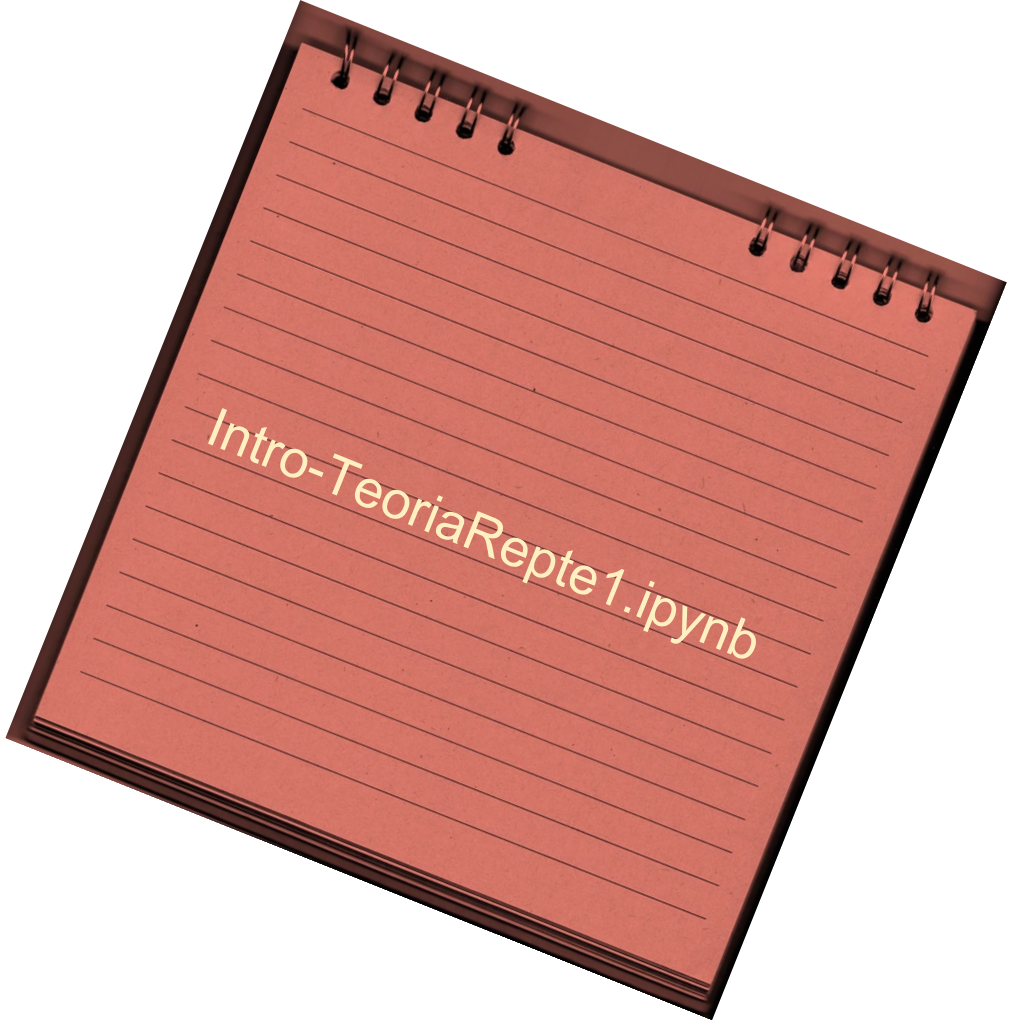
```python
edat: int = 20

if edat < 18:
    print("Menor d'edat")
elif edat < 65:  # opcionals, 0 o tants com vulguem
    print("Adult")
else:
    print("Sènior")
```

Ara et toca...

Escriu una funció que vagi demanant un número a l'usuari fins a que aquest entri 100.
Per a cada número que entri ha de mostrar un missatge a pantalla: has entrat un número molt petit (>=0 i <50), has entrat un número molt gran (>=50), has entrat un número negatiu (<0).
Quan l'usuari entri 100 ha de mostrar totes les desenes de 0 a 100, 100 inclòs, i ja no ha de demanar cap més número.

Més difícil...



Escriu una funció que retorni el factorial d'un número entrat per paràmetre. No usis cap llibreria de Python.

Ajuda de Python

- Quan no recordem alguna funció o mètode de Python podem consultar:
  - La documentació oficial del llenguatge
  - Els cursos de W3C
- ChatGPT ens pot ajudar però sempre cal verificar i entendre el què està fent.
  Als exàmens no es pot usar.

Ús de ChatGPT o altres IA generatives per al codi

La IA generativa actualment és un suport important en la programació professional

Pot ajudar en les fases inicials de programació, a depurar o explicar codi i a fer canvis de versions.

Ara bé, la generació de codi amb IA no és totalment fiable i si es fa servir cal saber interpretar els resultats i saber validar el codi generat.

És important doncs aprendre bé a programar i un cop se'n sap usar la IA com a suport.

En aquest primer curs, en el que encara s'està aprenent a programar, NO ES PERMET l'ús d'IA en cap prova avaluada.

Python: codi de ChatGPT 1 (amb errors!)

```python
def percentatge(part: float, total: float) -> float:
    """Retorna el percentatge que representa 'part'
 sobre 'total'"""
    return part / total * 100
```

Detecta l'error de la funció anterior
Què ha passat? El programa peta?
Sempre has de revisar el codi generat per IA.

Python: codi de ChatGPT 2 (amb errors!)

```python
def es_major_d_edat(edat: int) -> bool:
    """Comprova si una persona és major d'edat"""
    return edat > 18
```

Detecta l'error de la funció anterior
Què ha passat? El programa peta?
Sempre has de revisar el codi generat per IA.

Python: codi de ChatGPT 3 (amb errors!)

```python
def text_a_majuscules(text: str) -> str:
    """Converteix un text a majúscules"""
    text.upper()
    return text
```

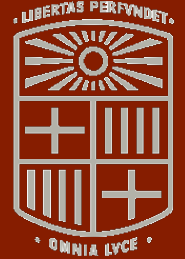
Detecta l'error de la funció anterior
Què ha passat? El programa peta? Upper modifica "text"?
Sempre has de revisar el codi generat per IA.

Notebooks de suport

Disposes de diversos notebooks per revisar i practicar el que hem vist fins ara: 3-EstilProgramacio, Lab1-0EntradaSortida.ipynb. Lab1-1RevEstructuresControl.ipynb, Tema1ConoceptesInicials-I.ipynb, Tema1ConceptesInicials-II.ipynb, Tema1FuncionsAccions.ipynb, PythonPractica1.ipynb

Gràcies i recapitulació

MIREIA RIBERA | ACCESSIBILITAT DIGITAL
| EXPERIÈNCIA D'USUARI
| VISUALITZACIÓ DE DADES



UNIVERSITAT DE BARCELONA

Recapitulació

✍️ En el codi usarem diferents noms i cal indiquin clarament que fan o què són.

💬 Els comentaris ajuden

⬅️ ➡️ A Python definim l'input amb els paràmetres i l'output per pantalla o com a retorn

🧮 Podem donar valor a una variable des d'un literal, una expressió o amb una funció.

🔄 Si volem repetir codi farem servir iteracions

🌿 Si volem executar una part de codi o una altra segons alguna variable o expressió farem servir condicionals.

🚫 🤖 No es permet l'ús de ChatGPT o altres IA a les proves d'avaluació.

Python II: Conceptes clau

1. Operadors relacionals i booleans
2. Cadenes de caràcters (strings)
3. Col·leccions de dades: llistes, diccionaris i tuples
4. Biblioteques externes

Python: operadors relacionals

A banda dels operadors aritmètics que ja heu vist, Python ens ofereix uns altres operadors que ens permeten comparar expressions i que donen com a resultat un valor booleà (*True* o *False*)

Són els operadors relacionals:

```
<       # menor que
<=      # menor o igual que
==      # igual que, no confondre amb l'assignació!
>       # major que
>=      # major o igual que
!=      # diferent a
```

Python: operadors booleans

A més, també podem combinar expressions lògiques amb els operadors booleans. De nou, el seu resultat és un valor booleà (*True* o False)

Són:

- and, *a and b*, el seu resultat és True si i només si els dos operands són True
- or, *a or b*, el seu resultat és True si com a mínim un dels dos operands és True
- not, *not a*, el seu resultat és invers al valor del seu operand.

Vegeu el notebook Lab2-Booleans.ipynb de Laboratori

Python: Operadors booleans

```python
a:bool = True
b:bool = False
c:bool = a and b
print(c)
print(a and b or c)
print(a or (not b) and c)
```

Prova de canviar els valors d'a, b i c i observa el resultat.

Python: cadenes de caràcters

Són un tipus de dades més complex perquè es guarda com un objecte, com una seqüència de caràcters indexats.

Aquesta estructura ens permet llegir un o més caràcters determinats amb els índexs.

```
a = 'Hola'
print(a)
type(a)
s = "Hello Bob"
x = 2
print(s[0], s[x], s[8-2])
> H l B
```

Vegeu el notebook Lab1-3Cadenes.ipynb de Laboratori

Python: cadenes de caràcters - immutables

Les cadenes de caràcters són immutables, és a dir no puc fer assignacions directes a cadascun dels seus caràcters.

```
# Les cadenes són immutables. El següent codi donarà error
paraula = "hola"
paraula[0] = "c"    # Error
```

Python: cadenes de caràcters i entrada

Quan demanem dades per teclat estem rebent strings i si volem tractar-los com a altres dades hem d'usar eval o fer una conversió explícita amb int, bool o float per exemple.

```
a: str = 'Hola'
print(a)
type(a)


nom = input("Quin és el teu nom?")
edat = eval(input("Quina és la teva edat?"))
```

Python: cadenes de caràcters - codificació

L'ordinador emmagatzema els caràcters com a nombres. A cadascun li assigna un codi. Inicialment els caràcters es van codificar amb la taula ASCII (American Standard Code for Information Interchange), però actualment es codifiquen amb la taula UniCode que inclou tots els idiomes del món.

```
print(chr(65))
print(ord("A"))
```

Python: cadenes de caràcters – mètodes i funcions

Python ens ofereix operadors, mètodes i funcions per facilitar el treball amb cadenes, els veureu amb més detall a les sessions de laboratori, aquí posem només alguns exemples:

```
print("jo" in "tu i jo")
text:str = "    Un exemple "
print(text[1:15:2])
llista: list[str] = text.split()
print(llista)
text_recuperat:str = " ".join(llista) print(text_recuperat)
print("text original:",text,". Amb strip:", text.strip(),". Però:",text)
print(text[4].isupper(),text[4].islower(),text[4].isalpha())
print(text.upper(),".Però:",text)
print(text.lower(),".Però:",text)
```

Ara et toca...

Escriu una funció que donat un text i una paraula indiqui si aquesta paraula es troba en el text (usa “in”). La funció ha de ser prou robusta per trobar paraules encara que estiguin amb accent o amb majúscules o minúscules.

Python: col·leccions de dades

Sovint els problemes necessiten treballar amb una col·lecció de “coses”, sovint no sabem quantes.

Per exemple: les dades d’un experiment, els estudiants d’un curs, els ingredients d’una recepta...

Ja hem vist una col·lecció, les llistes. Anem a veure algunes característiques importants

- Una llista és una seqüència ordenada de coses.
- Les llistes poden contenir qualsevol tipus de dades i en una mateixa llista hi poden haver diferents tipus. Són inhomogènies.
- Són mutables, és a dir, es poden canviar sobre la mateixa estructura.

Vegeu el notebook Lab1-4Colleccions.ipynb i Lab1-5Llistes.ipynb de Laboratori

Python: llistes – funcions i mètodes

Python ens ofereix operadors, mètodes i funcions per facilitar el treball amb llistes, els veureu amb més detall a les sessions de laboratori, aquí posem només alguns exemples:

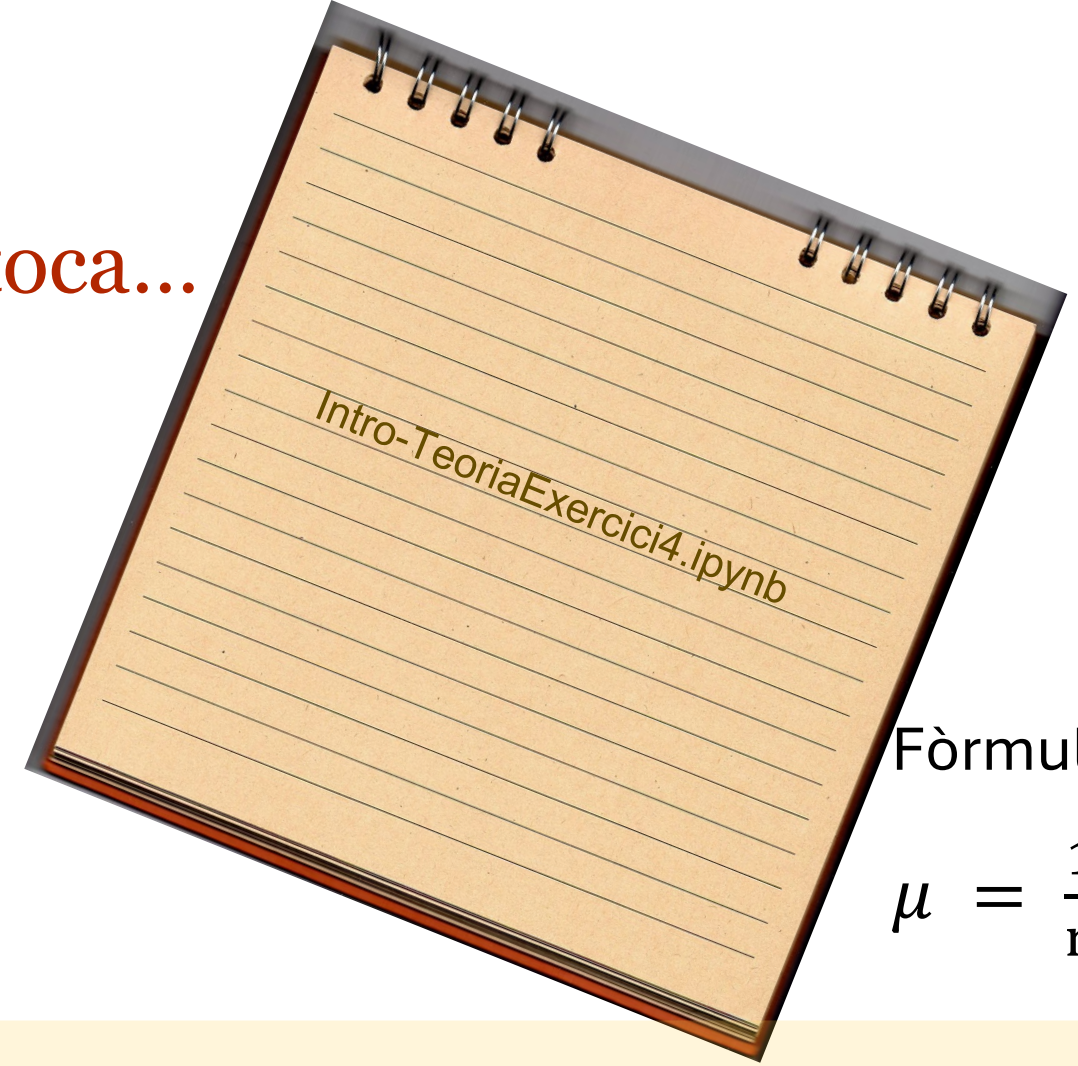
```
l1:list[Any] = []
l1.append("gat")
l1.append(2)
print(l1)
print("reverse:",l1.reverse(),l1)
l2:list[int] = [4, 2, 8, 9, 1, 2, 4, 7]
print("sort:",l2.sort(),l2)
print("count:",l2.count(4),l2.count(8))
print("index:",l2.index(9))
l2.remove(8)
print("Després del remove:",l2)
print("pop:",l2.pop(5))
print("Després del pop:",l2)
print("in:",9 in l2)
```

Python: matrius com a llistes de llistes

Python no té un tipus específic per a matrius, però les podem expressar com a llistes de llistes.

```
m = [[1,2,3],[4,5,6],[7,8,9]]
print(m[0])
print(m[1][1])
```

Ara et toca...



Fòrmules de la mitjana i desviació estàndard

$$\mu = \frac{1}{n} * \sum_{i=1}^{n} x_i \qquad \sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n-1}}$$

Escriu una funció que demani l'edat dels membres de la teva família o de la teva colla i que finalment mostri la mitjana (μ) i la desviació estàndard (σ).

Python: diccionaris

Els diccionaris són una altra col·lecció molt útil. Ens permeten guardar parelles de valors en les que un d'ells funciona com a índex i permet recuperar l'altre valor.

A laboratori veureu amb més detall el funcionament dels diccionaris, aquí posem només un exemple.

```
persones = {"33998877":"John Cooper","88996644":"Elizabeth Carl","34896745":"Erik
Peters"}

def digues_nom(dni:str) -> str:
    return persones[dni]

digues_nom("88996644")
>>'Elizabeth Carl'
```

Ara et toca...

Refés l'exercici 3 anterior, usant diccionaris per relacionar cada vocal amb accent amb la vocal corresponent sense accent.

Python: tuples

El darrer tipus de col·lecció que veurem són les tuples.

Les tuples són molt semblants a les llistes però són immutables. Això fa que el seu tractament informàtic sigui més eficient.

```
t: tuple = (1,2,3,4)
print(t[3])
print(t[2:4])
t1: tuple = ("gat",4,"gos")
print(t1)
t.append(5)
t[0]="canari"
```

Python: biblioteques externes

A banda dels tipus, mètodes i funcions bàsics de Python, aquest llenguatge disposa de múltiples biblioteques (en. *libraries*) externes que podem "importar" i usar. Hi ha biblioteques per treballar amb gràfics, per treballar amb dates...

En particular la biblioteca "math" ens serà força útil a l'assignatura.

```
import math

a:float = 4.5
b:int = 3
c:int = 2
d:int = 4

print(math.ceil(a))   # retorna l'enter més proper major que el nombre donat
print(math.cos(0.5)) # retorna  el cosinus. Math inclou totes les funcions trigonomètriques
print(math.sqrt(d))   # calcula l'arrel quadrada
```

Gràcies i recapitulació

MIREIA RIBERA | ACCESSIBILITAT DIGITAL
| EXPERIÈNCIA D'USUARI
| VISUALITZACIÓ DE DADES

UNIVERSITAT DE BARCELONA

Recapitulació

✅ ❌Hi ha operadors que retornen valors booleans

📝Les cadenes de caràcters i les llistes tenen un funcionament similar, però les cadenes són immutables.

📦 Cal triar la col·lecció que més ens convé: una llista, un diccionari o una tupla

🛠️Si volem una funció molt específica segurament hi ha una biblioteca externa que la inclou.