En aquest capítol aprendrem els elements bàsics del llenguatge de programació Python i per tant ja podrem escriure els nostres primers programes. És molt important, doncs, que tingueu ja instal·lat el Python (versió 3) i un bon editor de textos.

D'aquest capítol disposes dels següents arxius:

- El capítol en PDF: 02-Elements_basics-cat.pdf
- El notebook de Jupyter: cap2-cat.ipynb
- Els programes: programes2-cat.zip
- Enllaç al notebook a Google Colab: https://colab.research.google.com/github/aoliverg/python/blob/master/notebooks/cap2-cat.ipynb

2.1. Tipus bàsics, operacions i conversió entre tipus

2.1.a. Tipus bàsics

Un concepte molt important en informàtica és el de variable, que és un nom simbòlic associat amb un valor que pot canviar. Per exemple, si en Python fem:

```
a=3
```

estem assignant el valor 3 a la variable a. Les dades que poden emmagatzemar les variables poden ser de diferent tipus: nombres, cadenes, valors lògics, etc. Tot i que no és necessari declarar quin tipus de dada emmagatzema una variable (recordeu que en l'apartat 1.1. parlàvem del tipatge dinàmic com a una de les característiques de Python), un cop assignem un valor a una determinada variable, aquesta automàticament tindrà un tipus determinat que es mantindrà durant l'execució del programa, si no és que li assignem més endavant un valor d'un altre tipus.

Els tipus principals en Python són els següents:

- **Valors booleans** que poden prendre els valors Cert (True) o Fals (False).
- **Números** que poden ser sencers (integers) (1 i 2), de coma flotant (floats) (1.1 i 1.2), o complexos (complex) (3j+2)
- **Cadenes** (*strings*) són seqüències de caràcters unicode ("em dic Antoni" o "Меня зовут Антон")
- **Llistes** (*lists*), són seqüències ordenades de valors (["dilluns", "dimarts", "dimecres", "dijous", "divendres", "dissabte", "diumenge"].
- **Col·leccions** (*sets*) són un conjunt de valors sense ordre. El exemple anterior de la llista ens serviria, però tenint en compte que no tindrien un ordre determinat
- **Diccionaris** (*dictionaries*): són un conjunt sense ordre de parells clau-valor. Per exemple edat["Paula"]=27; edat["Joan"]=15; edat[Pau]=32, que també es podria representar com edat={'Joan': 15, 'Paula': 27, 'Pau': 32}

La funció type ens retorna el tipus d'un objecte. Ara obriu un intèrpret interactiu i escriviu les següents instruccions:

```
>>> a=True
>>> print(a)
True
>>> type(a)
<class 'bool'>
```

Fixeu-vos que type ens retorna el tipus de la variable. Com que en Python3 tot és una classe ens diu class

```
>>> a=3
>>> type(a)
```

<class 'int'>

Fixeu-vos que en la mateixa sessió de l'intèrpret interactiu heu tornat a fer servir la variable a per a un altres tipus i que s'ha canviat el tipus de la variable de manera dinàmica.

>>> a=1.1

>>> type(a)

<class 'float'>

>>> a=3j+2

>>> type(a)

<class 'complex'>

>>> a="em dic Antoni"

>>> type(a)

<class 'str'>

>>> a="Меня зовут Антон"

>>> type(a)

<class 'str'>

(si no podeu escriure en rus amb el teclat podeu provar de copiar la cadena a l'intèrpret)

>>> a=["dilluns", "dimarts", "dimecres", "dijous", "divendres", "dissabte", "diumenge"]

>>> type(a)

<class 'list'>

Podem fer coses de l'estil

>>> print(a[0])

dilluns

La primera posició és la 0

>>> print(a[1])

dimarts

I la segona la 1.

En aquest cas la darrera serà la 6, ja que tenim 7 elements.

>>> print(a[6])

diumenge

També podem accedir a la darrera posició amb l'índex -1:

```
>>> print(a[-1])
diumenge
```

Si intentem accedir a una posició no existent, es produeix un error:

```
>>> print(a[7])
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Ara convertirem la llista a en un set.

```
>>> a=set(a)
>>> type(a)
<class 'set'>
>>> print(a)
{'dissabte', 'dijous', 'dilluns', 'dimarts', 'dimecres', 'divendres', 'diumenge'}
```

Fixeu-vos que al passar a a set s'ha perdut l'ordre. A més si intentem accedir a una posició concreta es produeix un error:

```
>>> print(a[0])
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'set' object does not support indexing
```

Per començar a treballar amb un diccionari la millor manera és primer declarar-lo:

```
>>> a={}
```

I podem veure el tipus de la variable fent:

```
>>> type(a)
<class 'dict'>
```

Ara ja podem fer:

```
>>> edat["Paula"]=27
>>> edat["Joan"]=15
>>> edat["Pau"]=32
```

I si volem escriure tot el diccionari, fem:

```
>>> print(edat)
{'Joan': 15, 'Paula': 27, 'Pau': 32}
```

Però si volem accedir a una edat determinada fem:

```
>>> print(edat["Paula"])
27
```

Si intentem demanar el valor per a una clau no existent es produeix un error:

```
>>> print(edat["Antoni"])
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
KeyError: 'Antoni
```

Ara ja coneixem els principals tipus en Python. Cada un d'aquests tipus tindran associades una sèrie d'operacions possibles, però les anirem explicant a mesura que les anem necessitat en la resta de seccions.

2.1.b. Operacions bàsiques

Amb els tipus numèrics podem fer les operacions habituals, sumes, restes, etc.

```
>>> a=3
>>> b=2
>>> c=a+b
>>> print(c)
5
>>> d=a/b
>>> print(d)
1.5
```

Podem incrementar o decrementar el valor d'una variable de les següents maneres:

```
>>> a=1
>>> a=a+1
>>> print(a)
2
>>> a=1
>>> a+=1
>>> print(a)
2
```

Les cadenes les podem concatenar amb l'operador "+".

```
>>> a="hola"
>>> b="bon dia"
```

```
>>> d=a+" "+b
>>> print(d)
hola bon dia
```

Però compte, no podem fer servir l'operador "+" barrejant valors numèrics i cadenes.

```
>>> a="tinc"
>>> b=3
>>> c="pomes"
>>> d=a+" "+b+" "+c
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

Per poder fer-ho haurem de convertir el valor numèric en cadena (cosa que veurem en el següent subapartat, però que avancem ara):

```
>>> d=a+" "+str(b)+" "+c
>>> print(d)
tinc 3 pomes
```

2.1.c. Conversió entre tipus

Ja hem vist un exemple en el subapartat anterior de com convertir un valor numèric a cadena fent servir str(). Veiem ara les conversions més habituals:

De cadena a enter fent servir int() o a coma flotant fent servir float(). Recordeu que per cadenes l'operador "+" les concatena:

```
>>> a="3"
>>> b="2.1"
>>> print(a+b)
32.1
```

Si el que volem és obtenir la suma de 3 i 2.1 haurem de convertir-los en els valors numèrics corresponents:

```
>>> a=int(a)
>>> b=float(b)
>>> print(a+b)
5.1
```

També podríem haver convertit a directament a float i obtindríem el mateix resultat.

Recordeu l'exemple que ja hem vist de conversió de valor numèric a cadena fent servir str().

En el cas que tinguem una cadena amb una sèrie de valors separats per algun caràcter concret, podem convertir aquesta cadena en llista fent servir split(). Fixeu-vos que com a paràmetre a split li donem el separador:

```
>>> a="dilluns:dimarts:dimecres:dijous:divendres:dissabte:diumenge"
>>> b=a.split(":")
>>> print(b)
['dilluns', 'dimarts', 'dimecres', 'dijous', 'divendres', 'dissabte', 'diumenge']
```

De manera inversa, podem convertir una llista en una cadena ajuntant els seus elements per un separador concret amb join(), de la següent manera:

```
>>> c=",".join(b)
>>> print(c)
dilluns,dimarts,dimecres,dijous,divendres,dissabte,diumenge
```

Seria molt llarg i avorrit indicar totes les operacions i funcions disponibles en aquest subapartat Amb les que hem presentat serà suficient per començar a treballar en aplicacions reals. A mesura que les anem necessitant, anirem presentant noves operacions i funcions.

2.2. Control de flux

2.2.a. Introducció

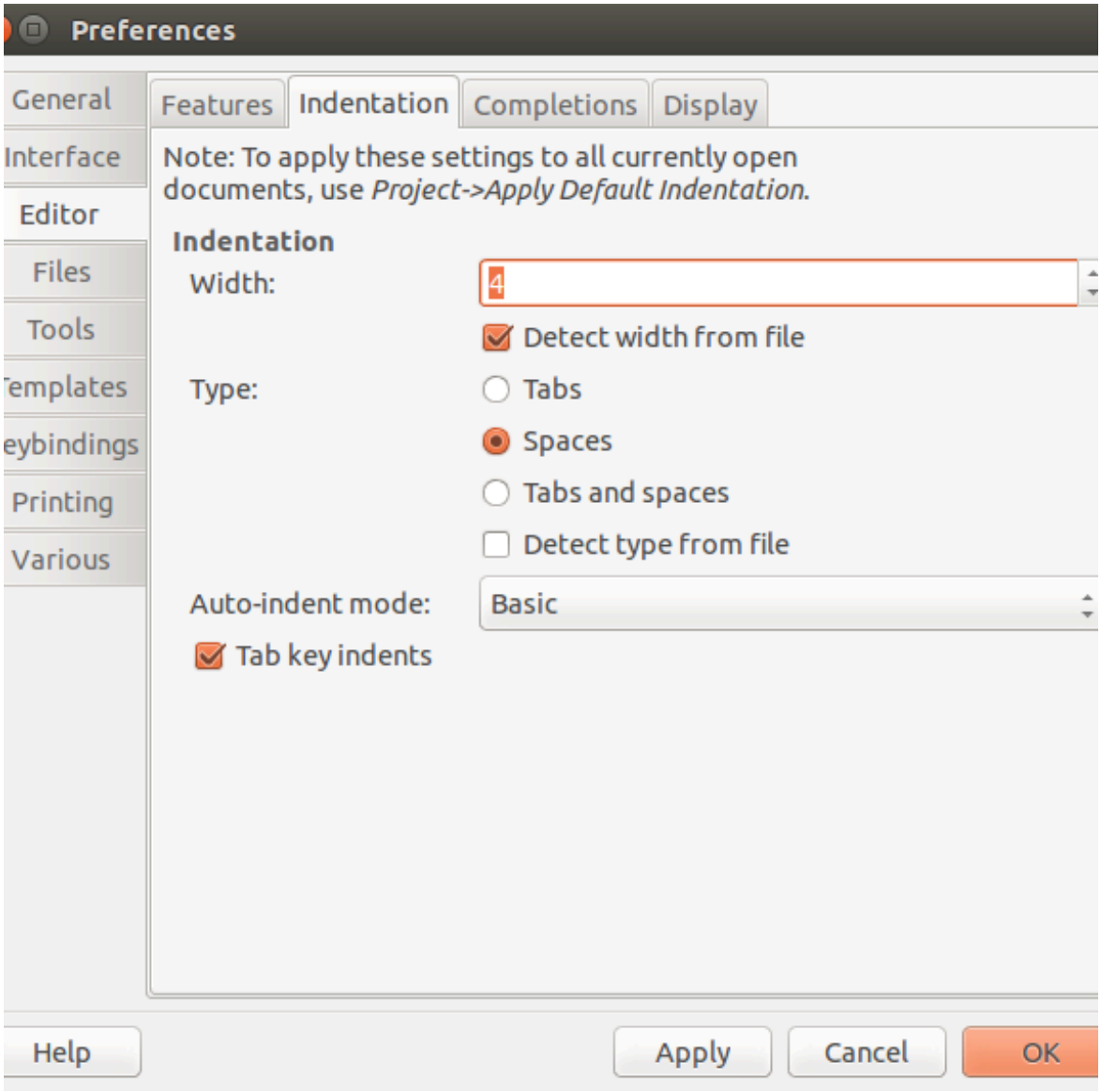
Els programes no són sempre una seqüència d'instruccions que s'executa una darrera de l'altra. Sovint, hi ha parts del codi que només s'executen si s'acompleix una condició determinada. També poden haver instruccions que s'executen repetidament fins que s'assoleix una determinada condició. En aquest apartat veurem les instruccions que ens permeten tenir el control sobre el flux d'execució del programa.

Un aspecte molt important a tenir en compte és que els blocs d'instruccions que depenen d'alguna instrucció control de flux es marquen mitjançant el sagnat del text, és a dir, deixant un nombre concret d'espais davant de les instruccions que depenen de la instrucció de control de flux. És molt important que el nombre d'espais sigui el mateix en tot el programa (habitualment 4 espais). També es pot substituir aquest nombre d'espais per un tabulador. Posem un exemple:

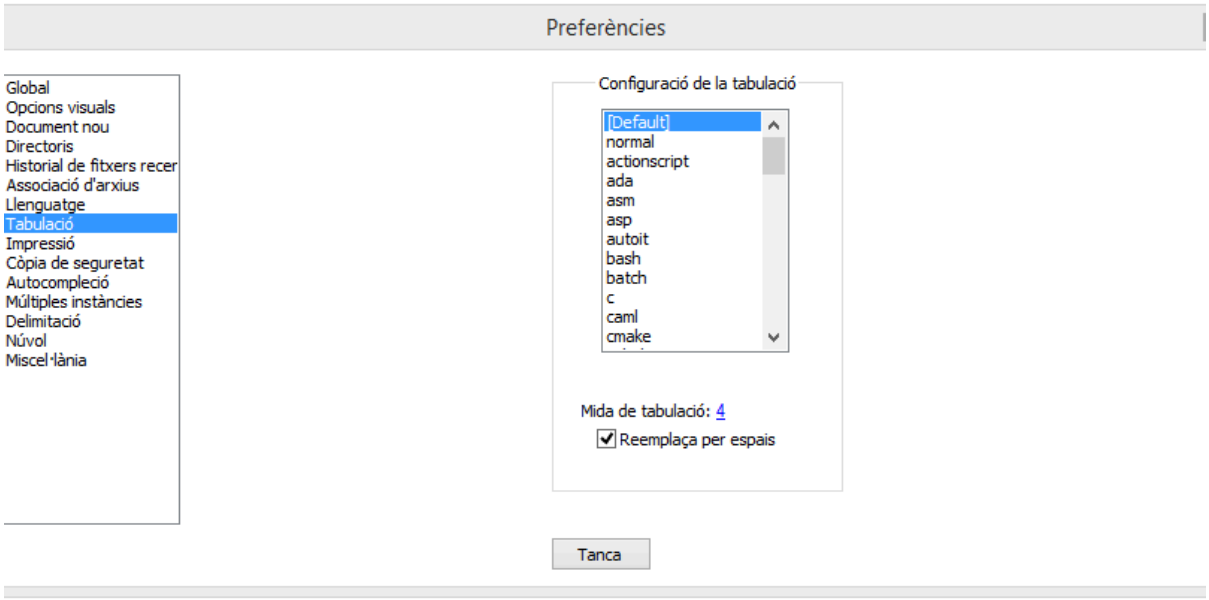
```
a=3
if a<5:
    print("Més petit que 5")
else:
    print("Més gran o igual que 5")
```

En aquest cas, el nombre d'espais que hi ha davant dels print és de 4. Aquest nombre d'espais s'haurà de mantenir durant tot el fitxer de programa. Donat que alguns dels programes els baixareu d'aquest espai i que haureu de fer modificacions, és important que ens posem d'acord en com ha de ser el sagnat en els nostres programes. Us proposo que sigui de 4 espais. Per fer-ho, en qualsevol editor de textos haureu de picar 4 espais seguits o bé configurar la tecla tabulador perquè escrigui 4 espais en lloc d'un tabulador. Per fer-ho:

En Geany: Aneu a **Edit > Preferences** i **Editor > Identation** i poseu les opcions com a la següent figura:



En Notepad++ aneu a **Configuració > Preferències** i seleccioneu **Tabulació**. Poseu les opcions com en la següent imatge:



En altres editors de text la configuració ha de ser similar.

2.2.b. Sentències condicionals: if, if...elif, if...elif...else

La sentència if ens permet indicar un condició i es pot complementar amb elif (sinó si) i else (sinó). Un exemple complet seria:

```
etiqueta="N"
if etiqueta=="N":
    print("Nom")
elif etiqueta=="V":
    print("Verb")
elif etiqueta=="A":
    print("Adjectiu")
elif etiqueta=="R":
    print("Adverbi")
else:
    print("Altres")
```

Si la condició només afecta a una instrucció no és necessari començar línia nova i això es pot escriure de manera molt més compacta com:

```
etiqueta="N"
if etiqueta=="N": print("Nom")
elif etiqueta=="V": print("Verb")
elif etiqueta=="A": print("Adjectiu")
```

```
elif etiqueta=="R": print("Adverbi")
else: print("Altres")
```

2.2.c. Bucles while

Un bucle és un conjunt d'instruccions que es repeteixen. Amb while podem fer que unes instruccions es repeteixen mentre s'acompleixi una condició. Un exemple senzill seria:

```
a=0
while a<=10:
    print(a)
    a+=1
```

La sortida d'aquest programa són els números del 0 al 10.

Si la condició s'acompleix sempre, creem un bucle infinit. Per exemple:

```
a=0
while 1:
    print(a)
    a+=1
```

Com que 1 és sempre 1, la condició es compleix indefinidament. Podem fer servir tambe while True: Aquestes construccions són habituals amb la instrucció break, com veurem una mica més endavant quan parlem dels fitxers, en la secció 2.3.

2.2.d. Bucles for

Serveix per iterar sobre els elements d'una seqüència, per exemple, una llista:

```
a=['dilluns', 'dimarts', 'dimecres', 'dijous', 'divendres', 'dissabte', 'diumenge']
for dia in a:
    print(dia)
```

Escriurà:

```
dilluns
dimarts
dimecres
dijous
divendres
dissabte
diumenge
```

La funció range() ens pot resultar molt útil per iterar sobre una seqüència de números. Per exemple:

```
for i in range(5):
    print
```

range(5) retorna una llista de 5 números, del 0 al 4, i d'aquesta manera la sortida del programa és:

```
0
1
2
3
4
```

range ens permet indicar l'element inicial, el final i el salt entre elements:

range(start, stop[, step])

Llavors:

```
for i in range(2,27,3):
    print
```

escriurà:

```
2
5
8
11
14
17
20
23
26
```

2.3. Treballar amb fitxers

2.3.a. Introducció

Molt sovint la informació que tractaran els nostres programes estarà emmagatzemada en arxius. Quan processem textos, serà necessari obrir-los, llegir les seves dades per processar-les i tancar-los. En aquesta secció veurem les instruccions bàsiques necessàries per fer les operacions més habituals amb arxius de text.

Com que els arxius de text poden estar en diferents codificacions de caràcters, ja des del principi ens acostumarem a treballar amb el mòdul codecs, que permet treballar molt fàcilment amb codificacions de caràcters.

2.3.b. Obrir i llegir un arxiu de text

Per obrir un arxiu de text en mode de lectura haurem de carregar el mòdul codecs i obrir l'arxiu de la següent manera:

```
import codecs

entrada=codecs.open("prova1.txt","r",encoding="utf-8")
```

Fixeu-vos que hem especificat el mode de lectura ("r") i que l'arxiu està en Unicode utf-8 ("utf-8").Els arxius es poden obrir en els següents modes:

- **r**: lectura
- **w**: escriptura
- **a**: per afegir dades al final de l'arxiu
- **r+**: tant per llegir com per escriure

A entrada ara tenim un objecte de tipus arxiu que té tres mètodes principals per llegir el contingut de l'arxiu:

- **read()**: llegeix tot l'arxiu
- **readline()**: llegeix una línia
- **readlines()**: llegeix totes les línies

Anem a practicar tot el relacionat amb la lectura d'arxius. Podeu descarregar un arxiu que es diu **arxiu.txt** i que està codificat un Unicode utf-8:

```
Aquesta es la primera linia.

Esta es la segunda línea.

This is the third line.

Это четвертая строка.

これは、5行目です。
```

A l'intèrpret interactiu podem provar les diverses opcions. Primer carreguem codecs i obrim l'arxiu en mode de lectura. Recordeu que si no esteu al directori on es trobar l'arxiu arxiu.txt, haureu d'indicar la ruta completa a l'arxiu.

```
import codecs

entrada=codecs.open("arxiu.txt","r",encoding="utf-8")
```

La primera opció que tenim és fer sevir read() que llegirà tot l'arxiu i el posarà en una cadena. Comprovem-ho:

```
>>> import codecs

>>> entrada=codecs.open("arxiu.txt","r",encoding="utf-8")

>>> contingut=entrada.read()

>>> print(contingut)

Aquesta es la primera línia.

Esta es la segunda línea.

This is the third line.

Это четвертая строка.

これは、5行目です。
```

```
>>> type(contingut)
<type 'unicode'>
```

Nota: és possible que en una pantalla de Símbol de sistema de Windows no es visualitzin correctament els caràcters ciríl·lics ni els japonesos. Es un problema de visualització, però la lectura es du a terme correctament.

Una altra opció és fer servir readline(), que et retorna una línia:

```
>>> import codecs
>>> entrada=codecs.open("arxiu.txt","r",encoding="utf-8")
>>> linia=entrada.readline()
>>> print(linia)

Aquesta es la primera línia.

>>> type(linia)
<type 'unicode'>
```

Si continuem llegint línies:

```
>>> linia=entrada.readline()
>>> print(linia)
Esta es la segunda línea.


>>> linia=entrada.readline()
>>> print(linia)
This is the third line.

>>> linia=entrada.readline()
>>> print(linia)
Это четвертая строка.

>>> linia=entrada.readline()
>>> print(linia)
これは、5行目です。
```

Si ja s'ha acabat l'arxiu i continuem:

```
>>> linia=entrada.readline()
>>> print(linia)

Ens retorna una cadena en blanc.
```

readline() sovint es fa servir dins d'un bucle infinit amb una condició que comprova si s'ha acabat l'arxiu. Això ho trobareu al programa programa-2-3-1.py.

```
import codecs
entrada=codecs.open("arxiu.txt","r",encoding="utf-8")
while 1:
    linia=entrada.readline()
    linia=linia.rstrip()
    if not linia:
        break
    print(linia)
entrada.close()
```

Aquesta solució es fa servir habitualment quan hem de tractar amb fitxers molt grans, ja que llegim una línia cada cop i evitem haver de carregar tot l'arxiu de text a memòria. Hem introduït rstrip(), que elimina els espais en blanc, inclosos els caràcters de salt de línia, del final de la cadena. També hem introduït **close()** que serveix per tancar l'arxiu un cop acabat de llegir-lo.

readlines() llegeix totes les línies de l'arxiu i l'emmagatzema un una llista. Veiem-ho amb l'intèrpret interactiu:

```
>>> entrada=codecs.open("arxiu.txt","r",encoding="utf-8")
>>> linies=entrada.readlines()
>>> print(linies)
[u'Aquesta es la primera l\xednia.\n', u'Esta es la segunda l\xednea.\n', u'This is the third line.\n', u'\u042d
\u0442\u043e \u0447\u0435\u0442\u0432\u0435\u0440\u0442\u0430\u044f \u0441\u0442\u0440\u043e\u043a
\u0430.\n', u'\u3053\u308c\u306f\u30015\u884c\u76ee\u3067\u3059\u3002\n']
>>> type(linies)
<type 'list'>
```

Fixeu-vos que no visualitzem correctament els caràcters no llatins bàsics, però si fem:

```
>>> entrada=codecs.open("arxiu.txt","r",encoding="utf-8")
>>> linies=entrada.readlines()
>>> for linia in linies:
... linia=linia.rstrip()
... print(linia)
...
Aquesta es la primera línia.
Esta es la segunda línea.
This is the third line.
Это четвертая строка.
これは、5行目です。
```

ho veiem correctament (potser sota Windows no). Encara tenim alguna altra opció per a llegir arxius de text:

Simplement iterar sobre el propi objecte arxiu:

```
>>> entrada=codecs.open("arxiu.txt","r",encoding="utf-8")
>>> for linia in entrada:
... linia=linia.rstrip()
... print(linia)
...
Aquesta es la primera línia.
Esta es la segunda línea.
This is the third line.
Это четвертая строка.
これは、5行目です。
```

Convertir l'objecte arxiu en una llista:

```
>>> entrada=codecs.open("arxiu.txt","r",encoding="utf-8")
>>> linies=list(entrada)
>>> print(linies)
[u'Aquesta es la primera l\xednia.\n', u'Esta es la segunda l\xednea.\n', u'This is the third line.\n', u'\u042d
\u0442\u043e \u0447\u0435\u0442\u0432\u0435\u0440\u0442\u0430\u044f \u0441\u0442\u0440\u043e\u043a
\u0430.\n', u'\u3053\u308c\u306f\u30015\u884c\u76ee\u3067\u3059\u3002\n']
```

2.3.c. Escriptura a arxius

Per escriure informació a un arxiu de text simplement cal fer servir write().

```
sortida=codecs.open("sortida.txt","w",encoding="utf-8")
>>> sortida.write("Escrivim una cadena de text.\n")
```

Si obriu l'arxiu sortida.txt veureu que conté:

```
Escrivim una cadena de text.
```

Fixeu-vos també que hem afegit manualment un salt de línia ("\n") al final de l'arxiu.

2.4. Errors i excepcions

2.4.1. Introducció

Quan programem segur que cometrem errors i fins i tot, programes correctament programats poden produir errors sota certes circumstàncies. Podem distingir dos tipus d'errors principals: errors de sintaxi i excepcions.

Els errors de sintaxi es produeixen perquè el codi conté algun tipus d'error i l'intèrpret el detecta i dóna un missatge d'error. A continuació observem un exemple:

```
>>> print("Hola)
File "<stdin>", line 1
print("Hola)
^
```

SyntaxError: EOL while scanning string literal

Fixeu-vos que falta tancar les cometes d'Hola i per això l'intèrpret avisa d'aquest error. El missatge d'error que proporciona l'intèrpret és de gran importància per corregir l'error: indica la línia on es produeix l'error i algun tipus d'explicació de l'error. Cal tenir en compte, però, que l'error pot estar en alguna línia anterior a la que indica l'intèrpret.

A continuació mostrem un exemple d'excepció (es tracta del programa-2-4-1.py)

```
a=input("Entra un número ")
b=input("Entra un altre numero ")
c=float(a)/float(b)
print("La divisió és ",c)
```

Aquest programa funciona perfectament en molts casos, però hi ha alguns en els que es produeix error.

Per exemple, per a 2 i b 3 funciona perfectament:

```
Entra un número 2
Entra un altre numero 3
La divisió és 0.6666666666666666
```

Ara, bé si per exemple al primer número no entrem un valor numèric, es produeix una excepció:

```
Entra un número w
Entra un altre numero 3
Traceback (most recent call last):
File "programa-2-4-1.py", line 3, in <module>
c=float(a)/float(b)
ValueError: could not convert string to float: 'w'
```

O bé si el segon número és un zero:

```
Entra un número 4
Entra un altre numero 0
Traceback (most recent call last):
File "programa-2-4-1.py", line 3, in <module>
c=float(a)/float(b)
ZeroDivisionError: float division by zero
```

Per evitar aquests errors podem escriure codi addicional de manera que controlem les possibles causes d'error, com per exemple el programa-2-4-2.py.

```
a=input("Entra un número ")
b=input("Entra un altre numero ")
```

```
if a.isnumeric() and b.isnumeric() and not b=="0":
    c=float(a)/float(b)
    print("La divisió és ",c)
else:
    print("Les xifres que has entrat no són correctes")
```

Proveu aquest programa i veureu que funciona prou bé però que només admet nombres sencers, ja que el mètode isnumeric() només verifica si tots els caràcters que conté la cadena són números. A la secció següent presentem una manera molt més efectiva de controlar els possibles errors que es poden produir a un programa.

2.4.2. try...except

Amb try...except podem controlar les excepcions que es produeixen a un programa. Observem a programa-2-4-3.py com podem controlar els errors d'aquesta manera.

```
a=input("Entra un número ")
b=input("Entra un altre numero ")
try:
    c=float(a)/float(b)
    print("La divisió és ",c)
except:
    print("Les xifres que has entrat no són correctes")
```

Podem fer que ens mostri el missatge d'error que es produeix important el mòdul sys i modificant la darrera línia (programa-2-4-4.py):

```
import sys
a=input("Entra un número ")
b=input("Entra un altre numero ")
try:
    c=float(a)/float(b)
    print("La divisió és ",c)
except:
    print("Les xifres que has entrat no són correctes",sys.exc_info()[0])
```