Programació per a la ciència de dades

Unitat 1: Estructures de dades avançades en Python

Instruccions d'ús ¶

Aquest document és un notebook interactiu que intercala explicacions més aviat teòriques de conceptes de programació amb fragments de codi executables. Per aprofitar els avantatges que aporta aquest format, us recomanem que, en primer lloc, llegiu les explicacions i el codi que us proporcionem. D'aquesta manera tindreu un primer contacte amb els conceptes que hi exposem. Ara bé, **la lectura és només el principi!** Una vegada hagueu llegit el contingut proporcionat, no oblideu executar el codi proporcionat i modificar-lo per crear-ne variants, que us permetin comprovar que heu entès la seva funcionalitat i explorar-ne els detalls d'implementació. Per últim, us recomanem també consultar la documentació enllaçada per explorar amb més profunditat les funcionalitats dels mòduls presentats.

In [1]:

```
%load_ext pycodestyle_magic
```

In [2]:

```
# Activem les alertes d'estil
%pycodestyle_on
```

Introducció

En aquesta unitat es presenten estructures de dades i tipus de dades avançats, i es fa una introducció a la manipulació avançada de cadenes de caràcters.

En primer lloc, s'expliquen estructures de dades avançades que amplien les estructures que ja coneixíem (llistes, tuples i diccionaris). Veurem què són i com fer servir en Python els conjunts, les piles, les cues, els diccionaris amb ordre i amb valors per defecte.

En segon lloc, veurem tipus de dades complexos. En concret, presentarem tipus de dades que permeten representar dates i hores, i explicarem les funcions que Python implementa sobre aquests tipus.

En tercer lloc i per acabar, ens centrarem en la manipulació avançada de cadenes de caràcters, repassant algunes de les funcions bàsiques sobre cadenes i introduint el llenguatge i ús d'expressions regulars.

A continuació s'inclou la taula de continguts, que podeu fer servir per a navegar pel document:

1.- Estructures de dades avançades

Fins ara hem vist algunes de les estructures de dades més bàsiques que es fan servir per programar en Python: les llistes, les tuples i els diccionaris. En aquesta unitat, presentarem estructures de dades més complexes, que ens permetran afrontar altres tipus de problemes, com són els conjunts, les piles, les cues, els diccionaris amb ordre, i els diccionaris amb valors per defecte.

1.1.- Conjunts

De la mateixa manera que les llistes, els conjunts són una col·lecció d'elements possiblement **heterogenis** (és a dir, que poden ser de tipus diferents). Ara bé, a diferència de les llistes, els elements d'un conjunt **no tenen ordre** (ni per tant posició dins del conjunt). A més, un conjunt **no pot tenir elements repetits**. En essència, aquesta estructura de dades implementa el concepte matemàtic de conjunt.

Python disposa de la classe `set` (https://docs.python.org/3.8/library/stdtypes.html#set-types-set-frozenset) que implementa el conjunt. Podem crear un conjunt fent servir el constructor de la classe o bé utilitzant claus `{}` :

In [3]:

```
# Creem un conjunt fent servir dues sintaxis diferents
a_set = {"Beale ciphers", "Quipu", "Zimmermann Telegram", "Chaocipher"}
the_same_set = set(
    ["Beale ciphers", "Quipu", "Zimmermann Telegram", "Chaocipher"])

# Confirmem que els dos conjunts creats són iguals
print("Boths sets are equal: {}".format(a_set == the_same_set))
```

```
Boths sets are equal: True
```

Cal anar en compte si fem servir claus per definir els conjunts, ja que aquestes també es fan servir per a definir diccionaris. Si els conjunts i diccionaris tenen elements, la sintaxi ens determinarà el tipus. En canvi, fer servir claus per definir una col·lecció buida resultarà en un diccionari buit:

In [4]:

```
# Definim dos diccionaris buits fent servir les dues sintaxis
an_empty_dict = {}
another_empty_dict = dict()
print("Using {{}} results in a {} equivalent to dict(): {}".
      format(type(an_empty_dict), an_empty_dict == another_empty_dict))

# Definim un conjunt buit
an_empty_set = set()
print("Creating an empty {}".format(type(an_empty_set)))
```

```
Using {} results in a <class 'dict'> equivalent to dict(): True
Creating an empty <class 'set'>
```

Els conjunts no tenen ordre i, per tant, dos conjunts són iguals si els seus elements també ho són:

In [5]:

```
# Creem un altre conjunt amb els mateixos elements, però fem servir un altre
# ordenació dels elements al crear el conjunt
another_same_set = {"Zimmermann Telegram", "Quipu",
                    "Chaocipher", "Beale ciphers"}
print("Boths sets are equal: {}".format(a_set == another_same_set))
```

```
Boths sets are equal: True
```

Com les llistes, els conjunts disposen de l'operació `len`, que retorna el número d'elements del conjunt (la seva cardinalitat). Les operacions `add` (https://docs.python.org/3.8/library/stdtypes.html#frozenset.add) i `remove` (https://docs.python.org/3.8/library/stdtypes.html#frozenset.remove) permeten afegir i eliminar elements d'un conjunt:

In [6]:

```
# Mostrem la cardinalitat del conjunt
print("The set is: {}".format(a_set))
print("The size of the set is: {}".format(len(a_set)))

# Eliminem un element del conjunt
a_set.remove("Quipu")
print("After removing 'Quipu', the set is: {}".format(a_set))

# Afegim un element al conjunt
a_set.add("The Magic Words are Squeamish Ossifrage")
print("After adding an element, the set is: {}".format(a_set))
```

```
The set is: {'Zimmermann Telegram', 'Quipu', 'Chaocipher', 'Beale ci
phers'}
The size of the set is: 4
After removing 'Quipu', the set is: {'Zimmermann Telegram', 'Chaocip
her', 'Beale ciphers'}
After adding an element, the set is: {'The Magic Words are Squeamish
Ossifrage', 'Chaocipher', 'Beale ciphers', 'Zimmermann Telegram'}
```

Els conjunts no tenen elements repetits:

In [7]:

```
print("The size of the set is: {}".format(len(a_set)))

# Afegim ara un element ja existent al conjunt
a_set.add("Beale ciphers")

# La cardinalitat del conjunt no varia, ja que l'element ja
# pertanyia al conjunt
print("The size of the set after adding 'Beale ciphers' is: {}"
      .format(len(a_set)))
```

```
The size of the set is: 4
The size of the set after adding 'Beale ciphers' is: 4
```

Ja hem vist com la classe `set` (https://docs.python.org/3.8/library/stdtypes.html#set) implementa la comparació d'igualtat de conjunts. Addicionalment, també implementa les operacions bàsiques de conjunts (unió, intersecció i diferència) i la diferència simètrica (que retorna els elements que estan només en un dels dos conjunts).

A continuació veiem un exemple d'aquestes operacions sobre tres conjunts, que contenen els noms dels estudiants de programació en Python, Java, i C, d'una escola (assumirem que el nom identifica de manera única els estudiants).

In [8]:

```
# Creem els tres conjunts
students_python = {"Anna", "Helena", "Marta", "Pol"}
students_java = {"Anna", "Teresa", "Helena"}
students_c = {"Marc"}

# Calculem els alumnes que estudien Python i també Java
print("Python and Java:\t\t{}".format(
    students_python.intersection(students_java)
))

# Calculem els alumnes que estudien Python però no java
print("Python but not Java:\t\t{}".format(
    students_python.difference(students_java)
))

# Calculem els alumnes que estudien Python o Java, però no
# tots dos llenguatges
print("Python or Java, but not both:\t{}".format(
    students_python.symmetric_difference(students_java)
))

# Calculem els estudiants que té l'escola (independentment
# del llenguatge que estudien)
print("Any of the languages:\t\t{}".format(
    students_python.union(students_java).union(students_c)
))
```

```
Python and Java:                {'Anna', 'Helena'}
Python but not Java:            {'Pol', 'Marta'}
Python or Java, but not both:   {'Marta', 'Teresa', 'Pol'}
Any of the languages:           {'Anna', 'Teresa', 'Marc', 'Helena',
'Pol', 'Marta'}
```

Noteu que els resultats de les operacions entre conjunts que hem vist a la cel·la anterior són també conjunts, de manera que podem encadenar operacions (com mostra l'últim exemple, on s'obtenen tots els estudiants de l'escola).

Equivalentment, podem fer servir els operands `|`, `&`, `-` i `^`:

In [9]:

```
print("Python and Java:\t\t{}".format(
    students_python & students_java))
print("Python but not Java:\t\t{}".format(
    students_python - students_java))
print("Python or Java, but not both:\t{}".format(
    students_python ^ students_java))
print("Any of the languages:\t\t{}".format(
    students_python | students_java | students_c))
```

```
Python and Java:                {'Anna', 'Helena'}
Python but not Java:            {'Pol', 'Marta'}
Python or Java, but not both:   {'Marta', 'Teresa', 'Pol'}
Any of the languages:           {'Anna', 'Teresa', 'Marc', 'Helena',
'Pol', 'Marta'}
```

Els conjunts són iterables, de manera que els podem recórrer com hem vist a la unitat anterior:

In [10]:

```
# Recorrem el conjunt students_python
for element in students_python:
    print(element)
```

```
Anna
Marta
Pol
Helena
```

Fins ara hem presentat les propietats dels conjunts i les operacions principals que podem realitzar sobre aquests amb la classe `set` (https://docs.python.org/3.8/library/stdtypes.html#set). A més, hem vist com els conjunts són l'estructura de dades a fer servir quan volem mantenir una col·lecció d'elements sense duplicats i on l'ordre no tingui importància. Els conjunts, així mateix, es poden fer servir com a eina per resoldre, de manera eficient, cert tipus de problemes.

D'una banda, els conjunts són una de les maneres més ràpides d'eliminar duplicats d'una llista:

In [11]:

```
# Definim una llista amb elements duplicats
a_list = ["Anna", "Helena", "Marta", "Pol", "Anna", "Teresa", "Helena"]
print("A list with duplicates:\t\t\t{}".format(a_list))

# Creem una nova llista sense duplicats, fent un cast de la llista inicial
# a conjunt, i convertint el resultat en llista de nou
a_list_without_duplicates = list(set(a_list))
print("The same list without duplicates:\t{}".format(
    a_list_without_duplicates))
```

```
A list with duplicates:                 ['Anna', 'Helena', 'Marta',
'Pol', 'Anna', 'Teresa', 'Helena']
The same list without duplicates:       ['Anna', 'Marta', 'Teresa',
'Pol', 'Helena']
```

A més d'eficient, la solució anterior és senzilla i elegant, seguint el que ens recomana el `PEP20` (https://www.python.org/dev/peps/pep-0020/).

Els conjunts en Python també es troben optimitzats per fer testos de pertinença, que ens permeten comprovar si un determinat element es troba present en un conjunt:

In [12]:

```
a_student = "Anna"

print("Python students:\t{}".format(students_python))
print("Anna studies Python?:\t{}\n".format(a_student in students_python))

print("C students:\t\t{}".format(students_c))
print("Anna studies C?:\t{}".format(a_student in students_c))
```

```
Python students:        {'Anna', 'Marta', 'Pol', 'Helena'}
Anna studies Python?:   True

C students:             {'Marc'}
Anna studies C?:        False
```

1.2.- Piles i cues

Les piles i les cues són unes estructures de dades també similars a les llistes, on els elements tenen ordre i poden estar repetits. La característica que defineix les piles i les cues és que els elements només es poden inserir i eliminar des d'un dels extrems.

Una **pila** (en anglès, *stack*) només permet afegir i esborrar elements al cim de la pila (en anglès, parlem del *top*), és a dir, per la part superior. Les piles es coneixen també com a estructures FILO (de l'anglès, *first in last out*), ja que el primer element que s'afegeix a una pila serà l'últim que en sortirà. Una analogia en el món físic d'una pila pot ser una pila de llibres que es troba dins d'una caixa: el primer llibre que traurem de la caixa serà el de dalt de tot de la pila, que serà l'últim que haguem posat a la caixa.

Una **cua** (en anglès, *queue*) només permet afegir elements al final i eliminar-ne del principi. Les cues es coneixen també com a estructures FIFO (de l'anglès, *first in first out*), ja que el primer element que s'afegeix a una cua serà el primer en sortir-ne. En el nostre dia a dia trobem cues en diverses circumstàncies, per exemple, quan esperem el nostre torn a la carnisseria o per comprar entrades al cinema.

A diferència dels conjunts, que en Python representem amb el tipus propi `set` (https://docs.python.org/3.8/library/stdtypes.html#set), per treballar amb piles i cues amb Python farem servir el tipus llista (`list` (https://docs.python.org/3.8/library/stdtypes.html#list)). Aquest tipus disposa d'un seguit d'operacions que permeten emular el comportament de piles i cues. Python també disposa d'una classe específica per gestionar cues, la classe `queue` (https://docs.python.org/3.8/library/queue.html), que es fa servir en implementacions *multithread* que veurem més endavant a l'assignatura.

Per a implementar una pila, treballarem sobre una llista, fent servir únicament els mètodes `append` i `pop` per afegir i treure-hi elements:

In [13]:

```
def print_stack(s, msg_bef=None, msg_after=None):
    # Definim la funció auxiliar print_stack, que ens permetrà
    # visualitzar una pila
    if msg_bef:
        print(msg_bef)

    if len(s) == 0:
        max_len = 10
    else:
        max_len = max([len(e) for e in s]) + 4

    print("|" + " "*(max_len+2) + "|")
    for e in s[::-1]:
        print("| " + e + " "*(max_len-len(e)) + " |")
    print("|" + "_"*(max_len+2) + "|")

    print(msg_after+"\n" if msg_after else "\n")
```

In [14]:

```
# Inicialitzem una llista buida, que farem servir com a pila
stack = []
print_stack(stack, "Initial stack:")

# Afegim 3 elements a la pila, i anem mostrant el contingut de
# la pila en cada moment
stack.append("The Art of Computer Programming")
print_stack(stack, "Stack after adding 1 element:")

stack.append("The Mythical Man-Month")
print_stack(stack, "Stack after adding a 2nd element:")

stack.append("Refactoring")
print_stack(stack, "Stack after adding a 3rd element:")
```

```
Initial stack:
|            |
|____________|


Stack after adding 1 element:
|                                      |
| The Art of Computer Programming      |
|______________________________________|


Stack after adding a 2nd element:
|                                      |
| The Mythical Man-Month               |
| The Art of Computer Programming      |
|______________________________________|


Stack after adding a 3rd element:
|                                      |
| Refactoring                          |
| The Mythical Man-Month               |
| The Art of Computer Programming      |
|______________________________________|
```

A l'afegir elements, aquests es van apilant de manera que l'últim element afegit queda al *top* de la pila. A l'exemple anterior, l'últim element afegit, *Refactoring*, queda al *top* de la pila.

In [15]:

```
# Eliminem un element de la pila i mostrem com queda la pila
e = stack.pop()
print_stack(stack, "Stack after deleting one element:",
            "The element removed was: {}".format(e))
```

```
Stack after deleting one element:
|                                        |
| The Mythical Man-Month                 |
| The Art of Computer Programming        |
|________________________________________|
The element removed was: Refactoring
```

A l'executar `pop`, s'elimina l'element del *top* de la pila, en aquest cas, *Refactoring*. L'últim element afegit és doncs el primer en sortir-ne.

In [16]:

```
# Eliminem dos elements més de la pila i mostrem com queda la pila
# en cada moment
e = stack.pop()
print_stack(stack, "Stack after deleting another element:",
            "The element removed was: {}".format(e))

e = stack.pop()
print_stack(stack, "Stack after deleting the last element:",
            "The element removed was: {}".format(e))
```

```
Stack after deleting another element:
|                                        |
| The Art of Computer Programming        |
|________________________________________|
The element removed was: The Mythical Man-Month

Stack after deleting the last element:
|             |
|_____________|
The element removed was: The Art of Computer Programming
```

Si seguim eliminant elements, veurem com el primer element afegit a la pila, *The Art of Computer Programming*, és l'últim a sortir-ne (el comportament FILO, *first in last out* que comentàvem).

Finalment, executar un `pop` sobre una pila buida, generarà una excepció:

In [17]:

```
try:
    stack.pop()
except IndexError as e:
    print(e)
```

```
pop from empty list
```

Per a implementar una cua, treballarem també sobre una llista, fent servir únicament els mètodes `append` i `pop(0)` per afegir i treure-hi elements:

In [18]:

```
def print_queue(s, msg_bef=None, msg_after=None):
    # Definim la funció auxiliar print_queue, que ens permetrà
    # visualitzar una cua
    if msg_bef:
        print(msg_bef)

    if len(s) == 0:
        max_len = 10
    else:
        max_len = sum([len(e)+2 for e in s]) + 1

    print("_"*(max_len+2)+"\n")
    print("  " + "  ".join(s))
    print("_"*(max_len+2))

    print(msg_after+"\n" if msg_after else "\n")
```

In [19]:

```
# Inicialitzem una llista buida, que farem servir com a cua
queue = []
print_queue(queue, "Initial queue:")

# Afegim 3 elements a la cua, i anem mostrant el contingut de
# la cua en cada moment
queue.append("A")
print_queue(queue, "Queue after adding 1 element:")

queue.append("B")
print_queue(queue, "Queue after adding a 2nd element:")

queue.append("C")
print_queue(queue, "Queue after adding a 3rd element:")
```

```
Initial queue:
_____________


_____________


Queue after adding 1 element:
______

  A
______


Queue after adding a 2nd element:
_________

  A  B
_________


Queue after adding a 3rd element:
_____________

  A  B  C
_____________
```

In [20]:

```
# Eliminem un element de la cua i mostrem com queda la cua
e = queue.pop(0)
print_queue(queue, "Queue after deleting one element:",
            "The element removed was: {}".format(e))
```

```
Queue after deleting one element:
_________

  B  C
_________
The element removed was: A
```

A l'executar `pop(0)` , s'elimina el primer element de la cua, en aquest cas, A, que és el primer element que hi hem afegit. Així doncs, la cua és una estructura FIFO (*first in first out*).

In [21]:

```
# Eliminem dos elements més de la cua i mostrem com queda la cua
# en cada moment
e = queue.pop(0)
print_queue(queue, "Queue after deleting another element:",
            "The element removed was: {}".format(e))

e = queue.pop(0)
print_queue(queue, "Queue after deleting the last element:",
            "The element removed was: {}".format(e))
```

```
Queue after deleting another element:
______

  C

______
The element removed was: B

Queue after deleting the last element:
____________


____________
The element removed was: C
```

L'últim element afegit és doncs també l'últim en sortir-ne (C).

De nou, si intentem eliminar un element sobre una cua buida, es genera una excepció:

In [22]:

```
try:
    queue.pop(0)
except IndexError as e:
    print(e)
```

```
pop from empty list
```

1.3.- Diccionaris ordenats

Tot i que a partir de la versió 3.6 de Python els diccionaris preserven l'ordre d'inserció dels elements, l'estructura de dades no contempla l'ordre dels elements a l'hora de fer comparacions. Així, dos diccionaris amb el mateix contingut inserit en ordre diferent són considerats iguals. És a dir, dos diccionaris són iguals si el seu contingut ho és, independentment de l'ordre en què s'hagi inserit aquest contingut:

In [23]:

```
# Creem dos diccionaris amb el mateix contingut en ordre diferent
dict_1 = {"a": 0, "b": 1}
dict_2 = {"b": 1, "a": 0}

# Mostrem el contingut dels diccionaris en l'ordre en què items() retorna
# els elements
print("dict_1:")
for k, v in dict_1.items():
    print("{}: {}".format(k, v))

print("\ndict_2:")
for k, v in dict_2.items():
    print("{}: {}".format(k, v))

# Comprovem la igualtat dels dos diccionaris
print("\ndict_1 and dict_2 are equal: {}".format(dict_1 == dict_2))
```

```
dict_1:
a: 0
b: 1

dict_2:
b: 1
a: 0

dict_1 and dict_2 are equal: True
```

Si pel contrari, volem que l'ordre dels elements d'un diccionari en determini la seva semblança a un altre diccionari, podem fer servir un diccionari ordenat, implementat per la classe `OrderedDict` (https://docs.python.org/3.8/library/collections.html#collections.OrderedDict) en Python, dins la llibreria `collections` (https://docs.python.org/3.8/library/collections.html).

In [24]:

```
from collections import OrderedDict

# Creem dos diccionaris ordenats amb el mateix contingut en ordre diferent
dict_1 = OrderedDict([("a", 0), ("b", 1)])
dict_2 = OrderedDict([("b", 1), ("a", 0)])

# Comprovem la igualtat dels dos diccionaris
print("\ndict_1 and dict_2 are equal: {}".format(dict_1 == dict_2))
```

```
dict_1 and dict_2 are equal: False
```

Més enllà de la diferència en la implementació de la igualtat, la classe `OrderedDict` (https://docs.python.org/3.8/library/collections.html#collections.OrderedDict) també ens ofereix un parell de mètodes relatius a l'ordre dels elements que no teníem disponibles en els diccionaris: `reversed` i `move_to_end` (https://docs.python.org/3.8/library/collections.html#collections.OrderedDict.move_to_end).

In [25]:

```
# Mostrem el contingut de dict_2 en l'ordre original
print("\ndict_2:")
for k in dict_2:
    print("{}: {}".format(k, dict_2[k]))

# Mostrem el contingut de dict_2 en ordre invers
print("\ndict_2:")
for k in reversed(dict_2):
    print("{}: {}".format(k, dict_2[k]))
```

```
dict_2:
b: 1
a: 0

dict_2:
a: 0
b: 1
```

In [26]:

```
# Creem un diccionari ordenat
dict_4 = OrderedDict([("a", 0), ("b", 1), ("c", 2), ("d", 3)])

# Movem l'element de clau 'b' al final del diccionari
dict_4.move_to_end('b')

# Mostrem el diccionari modificat, amb l'element 'b' al final
print(dict_4)
```

```
OrderedDict([('a', 0), ('c', 2), ('d', 3), ('b', 1)])
```

Com a apunt final, cal tenir en compte que en general un diccionari estàndard de Python és més eficient que un diccionari ordenat. Així doncs, farem servir sempre diccionaris estàndard, a no ser que necessitem les funcionalitats d'un diccionari ordenat per a la lògica de la nostra aplicació.

1.4.- Altres variants de diccionari

Una altra variant del diccionari que sovint és d'utilitat és el `defaultdict` (https://docs.python.org/3.8/library/collections.html#collections.defaultdict). Aquest tipus de diccionari comparteix les propietats del diccionari tradicional (`dict` (https://docs.python.org/3.8/library/stdtypes.html#mapping-types-dict)), però permet establir un valor per defecte a les entrades del diccionari no inicialitzades:

In [27]:

```
# Creem un diccionari a_trad_dict i intentem mostrar el valor d'una clau
# inexistent
a_trad_dict = {}
try:
    print(a_trad_dict["non_existing_key"])
except KeyError as e:
    print(e)
```

```
'non_existing_key'
```

In [28]:

```
from collections import defaultdict

# Creem un diccionari defaultdict amb valor per defecte una llista
# i intentem mostrar el valor d'una clau inexistent
a_defaultdict = defaultdict(list)
print(a_defaultdict["non_existing_key"])


def get_0():
    # Creem un diccionari defaultdict amb valor per defecte una funció
    # que retorna 0 i intentem mostrar el valor d'una clau inexistent
    return 0


another_defaultdict = defaultdict(get_0)
print(another_defaultdict["non_existing_key"])
```

```
[]
0
```

Els `defaultdict` (https://docs.python.org/3.8/library/collections.html#collections.defaultdict) poden ser útils per evitar haver d'inicialitzar les entrades d'un diccionari manualment.

Per exemple, suposem que tenim un text, i volem obtenir una llista de les paraules agrupades segons la seva lletra inicial. Podríem fer servir un diccionari per emmagatzemar aquesta agrupació: la clau del diccionari emmagatzemaria la lletra inicial, i el valor de cada clau seria una llista de les paraules que comencen per aquella lletra. Així, només caldria tenir entrades per les lletres que són inicials d'alguna de les paraules del text, i no per totes les possibles lletres inicials.

Una implementació bàsica podria fer servir un diccionari normal, processant cada paraula una a una i comprovant en cada cas si ja tenim una entrada al diccionari per la lletra inicial de la paraula:

In [29]:

```
words = "Beautiful is better than ugly. Explicit is better than implicit. "
words += "Simple is better than complex."

# Creem el diccionari
words_by_first_letter = dict()

# Iterem per cada paraula del text
for word in words.split(" "):
    # Recuperem la primera lletra
    first_letter = word[0]

    if first_letter in words_by_first_letter:
        # Si ja existeix l'entrada del diccionari, afegim la paraula
        # a la llista
        words_by_first_letter[first_letter].append(word)
    else:
        # Si no existeix encara cap paraula amb la mateixa inicial, creem
        # l'entrada al diccionari, creant la llista de paraules per aquella
        # inicial i afegint-hi la paraula
        words_by_first_letter[first_letter] = [word]

# Mostrem el resultat
print(words_by_first_letter)
```

```
{'B': ['Beautiful'], 'i': ['is', 'is', 'implicit.', 'is'], 'b': ['be
tter', 'better', 'better'], 't': ['than', 'than', 'than'], 'u': ['ug
ly.'], 'E': ['Explicit'], 'S': ['Simple'], 'c': ['complex.']}
```

Observem ara com podem simplificar el codi fent servir un `defaultdict` (https://docs.python.org/3.8/library/collections.html#collections.defaultdict):

In [30]:

```
# Creem el diccionari defaultdict
words_by_first_letter = defaultdict(list)

# Iterem per cada paraula del text
for word in words.split(" "):
    # Recuperem la primera lletra
    first_letter = word[0]
    # Afegim la paraula a la llista de la inicial corresponent
    words_by_first_letter[first_letter].append(word)

print(words_by_first_letter)
```

```
defaultdict(<class 'list'>, {'B': ['Beautiful'], 'i': ['is', 'is',
'implicit.', 'is'], 'b': ['better', 'better', 'better'], 't': ['tha
n', 'than', 'than'], 'u': ['ugly.'], 'E': ['Explicit'], 'S': ['Simpl
e'], 'c': ['complex.']})
```

Fixeu-vos que, en aquest cas, no ens cal inicialitzar manualment la llista.

2.- Tipus de dades avançats: el mòdul `datetime`

Ja coneixem els tipus de dades bàsics més habituals en Python: `int` (https://docs.python.org/3.8/library/functions.html#int) i `float` (https://docs.python.org/3.8/library/functions.html#float), que ens permeten representar valors numèrics; `bool` (https://docs.python.org/3.8/library/functions.html#bool), per a representar valors booleans; i `str` (https://docs.python.org/3.8/library/stdtypes.html#text-sequence-type-str), per representar cadenes de caràcters. També hem fet servir tipus compostos, com ara `list` (https://docs.python.org/3.8/library/stdtypes.html#list), `tuple` (https://docs.python.org/3.8/library/stdtypes.html#tuple), `dict` (https://docs.python.org/3.8/library/stdtypes.html#dict) i `set` (https://docs.python.org/3.8/library/stdtypes.html#set). En aquesta secció, presentarem altres classes que permeten representar tipus més complexos, com ara dates i temps.

El mòdul `datetime` (https://docs.python.org/3.8/library/datetime.html) de Python ofereix eines per treballar amb dates i hores. Així, aquest mòdul ens proveeix d'eines per afrontar les problemàtiques més habituals de la gestió de dates i hores, com són la representació del temps en cadenes de caràcters amb formats diferents, càlculs relatius a calendaris, càlcul d'increments de temps, conversions entre unitats de temps, gestió de fusos horaris, etc. Tot això facilita enormement la feina de tractar amb conjunts de dades que contenen dates i hores.

A tall d'exemple, intenteu pensar com implementaríeu, amb les llibreries de Python que hem presentat fins ara, petits programes que permetessin calcular:

1. A partir d'una data concreta, el dia de la setmana (dilluns, dimarts, ..., diumenge) que era, és o serà aquell dia.
2. Si els valors "2019-11-15" i "Friday, November 15, 2019" corresponen o no a la mateixa data.
3. Quin dia serà d'aquí a 30 dies.
4. Si l'any 2200 serà o no un any de traspàs.

Tot aquest tipus de problemes són fàcilment implementables amb les funcions que proporciona el mòdul `datetime` (https://docs.python.org/3.8/library/datetime.html). Les classes principals d'aquest mòdul són `date` (https://docs.python.org/3.8/library/datetime.html#date-objects), que permet treballar amb dates; `time` (https://docs.python.org/3.8/library/datetime.html#time-objects) amb temps; `datetime` (https://docs.python.org/3.8/library/datetime.html#datetime-objects) amb dates completes que inclouen també el temps; i `timedelta` (https://docs.python.org/3.8/library/datetime.html#timedelta-objects) que processa increments de temps.

2.1.- Creació d'objectes que representen dates i hores

Podem crear objectes que representen dates, hores o dates amb hores especificant les diferents components de cada estructura manualment. L'exemple següent mostra com crear una data, una hora i una data amb hora fent servir aquesta estratègia:

In [31]:

```
import datetime

# Creem un objecte date que representa una data concreta
a_date = datetime.date(year=2017, month=10, day=1)
# De manera més concisa, podríem fer:
# a_date = datetime.date(2017, 1, 11)
print(a_date)

# Creem un objecte time que representa una hora
a_time = datetime.time(hour=17, minute=14, second=42)
# De manera més concisa, podríem fer:
# a_time = datetime.time(17, 14, 42)
print(a_time)

# Creem un objecte datetime, que representa una data amb temps
a_datetime = datetime.datetime(year=2017, month=10, day=1,
                               hour=17, minute=14, second=42)
# De manera més concisa, podríem fer:
# a_datetime = datetime.datetime(2017, 1, 11, 17, 14, 42)
print(a_datetime)
```

```
2017-10-01
17:14:42
2017-10-01 17:14:42
```

Ara bé, és molt habitual trobar conjunts de dades en què les dates i hores no es troben pas desglossades en any, mes, dia, hora, minut i segon, sinó que aquestes es representen com a cadenes de caràcters, que poden seguir convencions diferents a l'hora d'expressar les dates i les hores. Així, ens caldrà poder crear els objectes a partir de cadenes de caràcters, i necessitarem algun llenguatge per poder definir el format d'aquestes cadenes.

Les classes `datetime` (https://docs.python.org/3.8/library/datetime.html), `date` (https://docs.python.org/3.8/library/datetime.html#date-objects) i `time` (https://docs.python.org/3.8/library/datetime.html#time-objects) disposen del mètode `strptime` (https://docs.python.org/3.8/library/datetime.html#datetime.datetime.strptime) que permet crear un objecte `datetime` (https://docs.python.org/3.8/library/datetime.html), `date` (https://docs.python.org/3.8/library/datetime.html#date-objects) i `time` (https://docs.python.org/3.8/library/datetime.html#time-objects) a partir d'una cadena de caràcters amb la data i/o hora i d'una altra cadena de caràcters que n'informa del format. A continuació s'exposa com crear un objecte `datetime` (https://docs.python.org/3.8/library/datetime.html) a partir d'una cadena de caràcters. Tota aquesta funcionalitat pot ser utilitzada també de manera anàloga per a crear objectes `date` (https://docs.python.org/3.8/library/datetime.html#date-objects) i `time` (https://docs.python.org/3.8/library/datetime.html#time-objects).

Així doncs, podem crear un objecte `datetime` (https://docs.python.org/3.8/library/datetime.html) a partir de la cadena de caràcters "2017-10-01 17:14:42" de la manera següent:

In [32]:

```
str_date = "2017-10-01 17:14:42"
a_datetime_from_str = datetime.datetime.strptime(str_date, "%Y-%m-%d %H:%M:%S")
print(a_datetime_from_str)
a_datetime == a_datetime_from_str
```

```
2017-10-01 17:14:42
```

Out[32]:

```
True
```

El primer paràmetre de la crida, "2017-10-01 17:14:42", informa de la data a representar, mentre que el segon paràmetre, "%Y-%m-%d %H:%M:%S", informa del format que fa servir la primera cadena. Per representar el format, es fan servir uns comodins, indicats amb el símbol `%` , i altres caràcters no especials (com ara el guionet, `-` o els dos punts, `:`). Així, en aquest cas, informem que la cadena que representa la data i el temps està formada per:

- l'any representat fent servir quatre dígits (`%Y`),
- un guió que separa l'any del mes (`-`) ,
- el mes representat fent servir dos dígits (`%m`),
- un guió que separa el mes del dia (`-`),
- el dia representat fent servir dos dígits (`%d`),
- un espai separant la data de l'hora ( ),
- l'hora representada en format 24 hores i fent servir dos dígits (`%H`),
- els dos punts separant l'hora dels minuts (`:`),
- els minuts representats amb dos dígits (`%M`),
- els dos punts separant els minuts dels segons (`:`),
- i finalment, els segons representats amb dos dígits (`%S`).

És important notar que l'especificació del format ens permet evitar cap mena d'ambigüitat a l'hora d'interpretar la data. Així, per exemple, podem saber que es vol representar el dia 1 d'Octubre i no pas el dia 10 de Gener. Podríem canviar aquesta interpretació, modificant la cadena que expressa el format, de manera que el mes i el dia s'intercanviessin l'ordre:

In [33]:

```
a_date_from_str = datetime.datetime.strptime(str_date, "%Y-%d-%m %H:%M:%S")
print(a_date_from_str)
```

```
2017-01-10 17:14:42
```

Fixeu-vos que, a l'exemple anterior, la cadena que representa la data és exactament la mateixa, però canviant l'especificació del format, hem fet que s'interpreti de manera diferent, indicant ara el 10 de gener.

La cadena de caràcters que hem fet servir sempre utilitza el format complet per expressar qualsevol unitat de temps, anteposant zeros si cal. Així, per representar el primer dia del mes, fa servir dos dígits, `01` . Existeixen comodins per representar també les diferents unitats sense anteposar el zero, així com per representar anys fent servir només dos dígits, mesos fent servir el nom del mes en comptes del número, hores en format AM/PM, i moltes altres variants! Podeu trobar l'especificació completa del format a la documentació oficial del mòdul `datetime` (https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes). A continuació, s'exposen alguns exemples addicionals:

In [34]:

```
# Canviant el format dels separadors
a_date_from_str = datetime.datetime.strptime(
    "2017/10/01 17*14*42", "%Y/%m/%d %H*%M*%S")
print(a_date_from_str)
```

```
2017-10-01 17:14:42
```

In [35]:

```
# Expressant l'any amb dos dígits
a_date_from_str = datetime.datetime.strptime(
    "17-10-01 17:14:42", "%y-%m-%d %H:%M:%S")
print(a_date_from_str)
a_date_from_str = datetime.datetime.strptime(
    "95-10-01 17:14:42", "%y-%m-%d %H:%M:%S")
print(a_date_from_str)
```

```
2017-10-01 17:14:42
1995-10-01 17:14:42
```

Noteu com, en el cas anterior, Python interpreta el segle en funció dels últims dos dígits de l'any.

In [36]:

```
# Incloent el dia de la setmana i expressant el mes amb el nom
a_date_from_str = datetime.datetime.strptime(
    'Sunday, 01 October 2017 17:14:42', "%A, %d %B %Y %H:%M:%S")
print(a_date_from_str)
```

```
2017-10-01 17:14:42
```

In [37]:

```
# Amb el dia de la setmana i el mes abreviats
a_date_from_str = datetime.datetime.strptime(
    'Sun, 01 Oct 2017 17:14:42', "%a, %d %b %Y %H:%M:%S")
print(a_date_from_str)
```

```
2017-10-01 17:14:42
```

En informàtica, també es pot fer servir el *unix time stamp* com a manera de mesurar el temps. El *unix time stamp* representa el temps com a un enter, que compta els segons que han passat des de l'1 de Gener de 1970. Podem crear objectes que representen una data a partir d'un *unix timestamp* amb el mètode `fromtimestamp` (https://docs.python.org/3.8/library/datetime.html#datetime.date.fromtimestamp):

In [38]:

```
a_date_from_ts = datetime.date.fromtimestamp(1573840426)
print(a_date_from_ts)
```

```
2019-11-15
```

Sovint també ens interessarà fer servir la data o temps actual en les nostres aplicacions. Podem crear objectes que representin el moment actual a través de les funcions `today` (https://docs.python.org/3.8/library/datetime.html#datetime.datetime.today) i `now` (https://docs.python.org/3.8/library/datetime.html#datetime.datetime.now):

In [39]:

```
# Creem un objecte amb la data actual
now_date = datetime.date.today()
print(now_date)

# Creem un objecte amb la data i hora actual
now_datetime = datetime.datetime.now()
print(now_datetime)
```

```
2020-03-10
2020-03-10 18:40:51.590807
```

2.2.- Mostrant la data i el temps en diferents formats

Fins ara hem vist com el mètode `strptime` (https://docs.python.org/3.8/library/datetime.html#datetime.datetime.strptime) permet crear objectes `datetime` (https://docs.python.org/3.8/library/datetime.html), `date` (https://docs.python.org/3.8/library/datetime.html#date-objects) i `time` (https://docs.python.org/3.8/library/datetime.html#time-objects) a partir d'una cadena de caràcters. De manera anàloga, el mètode `strftime` (https://docs.python.org/3.8/library/datetime.html#datetime.datetime.strftime) permet obtenir una cadena de caràcters que representa una data, hora, o data amb hora en un format concret, que s'especifica fent servir la mateixa sintaxis que `strptime` (https://docs.python.org/3.8/library/datetime.html#datetime.datetime.strptime).

In [40]:

```
# Creem un objecte amb la data i hora actual
now_datetime = datetime.datetime.now()

# Mostrem la data i hora fent servir 4 dígits pels anys, i 2
# per als mes, dia, hora, minut i segons
print(now_datetime.strftime("%Y-%m-%d %H:%M:%S"))

# Invertim l'ordre del dia i el mes
print(now_datetime.strftime("%Y-%d-%m %H:%M:%S"))

# Canviant el format dels separadors
print(now_datetime.strftime("%Y/%m/%d %H*%M*%S"))

# Expressant l'any amb dos dígits
print(now_datetime.strftime("%y-%m-%d %H:%M:%S"))

# Incloent el dia de la setmana i expressant el mes amb el nom
print(now_datetime.strftime("%A, %d %B %Y %H:%M:%S"))

# Amb el dia de la setmana i el mes abreviats
print(now_datetime.strftime("%a, %d %b %Y %H:%M:%S"))
```

```
2020-03-10 18:40:51
2020-10-03 18:40:51
2020/03/10 18*40*51
20-03-10 18:40:51
Tuesday, 10 March 2020 18:40:51
Tue, 10 Mar 2020 18:40:51
```

A més de fer servir `strftime` (https://docs.python.org/3.8/library/datetime.html#datetime.datetime.strftime) per obtenir cadenes de caràcters que representin dates, també ens pot ser d'utilitat la funció `isoformat` (https://docs.python.org/3.8/library/datetime.html#datetime.date.isoformat), que ja retorna la cadena en format ISO 8601 (any - mes - dia, fent servir 4 dígits per a l'any i 2 per al dia i mes), sense necessitat d'especificar-lo manualment:

In [41]:

```
# Mostrem la data i l'hora en format ISO 8601
now_datetime.isoformat()
```

Out[41]:

```
'2020-03-10T18:40:51.602326'
```

2.3.- Treballant amb dates i temps

Ja hem vist una de les funcionalitats dels objectes que representen dates i temps, que és la d'interpretar i generar cadenes de caràcters que representen dates en diferents formats. A continuació, veurem altres operacions habituals sobre aquests objectes.

Podem accedir a les diferents components que conformen una data amb hora consultant els atributs dels objectes `datetime` (https://docs.python.org/3.8/library/datetime.html), `date` (https://docs.python.org/3.8/library/datetime.html#date-objects) i `time` (https://docs.python.org/3.8/library/datetime.html#time-objects). Noteu que, en aquest cas, el tipus de dades dels valors als que accedim és un enter, en comptes d'una cadena de caràcters com obteníem amb `strftime` (https://docs.python.org/3.8/library/datetime.html#datetime.datetime.strftime):

In [42]:

```
# Accedim a les diferents components d'una data amb hora
y = now_datetime.year
mo = now_datetime.month
d = now_datetime.day
h = now_datetime.hour
mi = now_datetime.minute
s = now_datetime.second

print("Year is {} and is an {}".format(y, type(y)))
print("Month is {} and is an {}".format(mo, type(mo)))
print("Day is {} and is an {}".format(d, type(d)))
print("Hour is {} and is an {}".format(h, type(h)))
print("Minute is {} and is an {}".format(mi, type(mi)))
print("Second is {} and is an {}".format(s, type(s)))
```

```
Year is 2020 and is an <class 'int'>
Month is 3 and is an <class 'int'>
Day is 10 and is an <class 'int'>
Hour is 18 and is an <class 'int'>
Minute is 40 and is an <class 'int'>
Second is 51 and is an <class 'int'>
```

Més enllà de l'accés a les components que conformen la data, també podem accedir a altres dades d'aquesta, com ara a quina setmana de l'any pertany o bé a quin dia de la setmana correspon:

In [43]:

```
# Obtenim una tupla de 3 valors: (any ISO, núm. de setmana ISO,
# dia de la setmana ISO)
iso_cal = now_datetime.isocalendar()
print("The week is: {}".format(iso_cal[1]))
print("The day is: {}".format(iso_cal[2]))
```

```
The week is: 11
The day is: 2
```

In [44]:

```
# També podem obtenir únicament el dia de la setmana amb weekday()
print("The day is: {}".format(now_datetime.weekday()))
```

```
The day is: 1
```

Les dues alternatives fan servir convencions diferents per referir-se als dies de la setmana: `weekday` (https://docs.python.org/3.8/library/datetime.html#datetime.datetime.weekday) considera que el Dilluns és un 0 i el Diumenge és un 6, mentre que `isocalendar` (https://docs.python.org/3.8/library/datetime.html#datetime.datetime.isocalendar) representa els dilluns amb un 1 i els diumenges amb 7.

Per a convertir el valor numèric retornat per `weekday` (https://docs.python.org/3.8/library/datetime.html#datetime.datetime.weekday) a una cadena de caràcters amb el nom del dia de la setmana, podem fer servir el mòdul `calendar` (https://docs.python.org/3.8/library/calendar.html):

In [45]:

```
# Importem el mòdul calendar
import calendar

# Obtenim el nom del dia
print('Day of Week: ', calendar.day_name[now_datetime.weekday()])

# Alternativament, també hagués sim pogut fer servir strftime per obtenir la
# mateixa dada
print(now_datetime.strftime("Short day name: %a"))
print(now_datetime.strftime("Full day name: %A"))
```

```
Day of Week:  Tuesday
Short day name: Tue
Full day name: Tuesday
```

La gestió de l'idioma en què es mostren els noms dels dies de la setmana o dels mesos es configura amb el mòdul `locale` (https://docs.python.org/3.8/library/locale.html). Així, per exemple, si volem obtenir el nom del dia de la setmana en català, faríem:

In [46]:

```
# Importem el mòdul locale
import locale

# Especifiquem l'idioma català
locale.setlocale(locale.LC_ALL, 'ca_ES.UTF-8')

# Mostrem el nom del dia
print('Day of Week:', calendar.day_name[now_datetime.weekday()])
```

```
Day of Week: dimarts
```

Una altra de les característiques que implementen els objectes de representació de dates i hores que treballem en aquesta unitat és la possibilitat de comparar-los:

In [47]:

```
# Importem la funció sleep, que permet introduir esperes en l'execució de codi
from time import sleep

# Obtenim la data i hora actuals
now_1_dt = datetime.datetime.now()
# Esperem 3 segons
sleep(3)
# Obtenim la data i hora actuals (després de l'espera de 3 segons)
now_2_dt = datetime.datetime.now()

# Comparem les dues dates
print('now_1_dt == now_2_dt: {}'.format(now_1_dt == now_2_dt))
print('now_1_dt < now_2_dt: {}'.format(now_1_dt < now_2_dt))
print('now_1_dt.date() == now_2_dt.date(): {}'.format(
    now_1_dt.date() == now_2_dt.date()))
```

```
now_1_dt == now_2_dt: False
now_1_dt < now_2_dt: True
now_1_dt.date() == now_2_dt.date(): True
```

Els dos valors de data i hora no són iguals (ja que hi ha tres segons de diferència) però, en canvi, el dia sí que és el mateix (a no ser que executem el codi just en el moment en què canviem de dia).

2.4.- Increments de temps

Una altra de les classes que incorpora el mòdul `datetime` (https://docs.python.org/3.8/library/datetime.html) és `timedelta` (https://docs.python.org/3.8/library/datetime.html#timedelta-objects). Aquesta classe ens permet representar increments de temps, així com manipular dates i hores utilitzant aquests increments.

Fixem-nos, per exemple, en les dues dates que hem creat anteriorment amb diferència de tres segons. Hem vist que les podíem comparar, per saber quina és anterior o si són iguals. Ara bé, com podríem saber el temps que ha transcorregut entre una i l'altre? Podríem calcular-ho manualment a partir d'anar recuperant les diferents components de la data amb hora, però la classe `timedelta` (https://docs.python.org/3.8/library/datetime.html#timedelta-objects) ja implementa aquesta funcionalitat:

In [48]:

```
# Calculem el temps transcorregut i el mostrem
delta = now_2_dt - now_1_dt
print("Datetimes differ in: {} seconds".format(delta))
print("Delta is: {}".format(type(delta)))
```

```
Datetimes differ in: 0:00:03.003140 seconds
Delta is: <class 'datetime.timedelta'>
```

El resultat de restar dos objectes `datetime` (https://docs.python.org/3.8/library/datetime.html#datetime-objects) és un objecte `timedelta` (https://docs.python.org/3.8/library/datetime.html#timedelta-objects) que representa el temps transcorregut. A continuació, veurem alguns exemples més de com es fan servir objectes `timedelta` (https://docs.python.org/3.8/library/datetime.html#timedelta-objects) per representar increments de temps:

In [49]:

```
# Creem tres datetimes
dt_1 = datetime.datetime(year=2017, month=10, day=1,
                         hour=17, minute=14, second=42)

dt_2 = datetime.datetime(year=2017, month=12, day=1,
                         hour=17, minute=14, second=42)

dt_3 = datetime.datetime(year=2017, month=10, day=15,
                         hour=20, minute=16, second=42)

# Creem timedeltas amb les diferències
tm_1 = dt_2 - dt_1
tm_2 = dt_3 - dt_1
tm_3 = dt_3 - dt_2
print("Timedelta between dt_1 and dt_2: {}".format(tm_1))
print("Timedelta between dt_1 and dt_3: {}".format(tm_2))
print("Timedelta between dt_2 and dt_3: {}".format(tm_3))
```

```
Timedelta between dt_1 and dt_2: 61 days, 0:00:00
Timedelta between dt_1 and dt_3: 14 days, 3:02:00
Timedelta between dt_2 and dt_3: -47 days, 3:02:00
```

Fixeu-vos com al mostrar per pantalla el valor d'un `timedelta` (https://docs.python.org/3.8/library/datetime.html#timedelta-objects), obtenim una cadena de caràcters anotada (s'inclou el número de dies i la paraula `days`, seguida de l'hora informada amb hora:minut:segons).

Si ho preferim, també podem obtenir el número de segons que representa el `timedelta` (https://docs.python.org/3.8/library/datetime.html#timedelta-objects) com a valor numèric (en aquest cas, un `float` (https://docs.python.org/3.8/library/functions.html#float)):

In [50]:

```
# Mostrem el número de segons del tm_1
ts = tm_1.total_seconds()
print("Seconds elapsed: {}".format(ts))
print("Result is: {}".format(type(ts)))
```

```
Seconds elapsed: 5270400.0
Result is: <class 'float'>
```

També podem construir un objecte `timedelta` (https://docs.python.org/3.8/library/datetime.html#timedelta-objects) per obtenir dates passades o futures a partir d'una data de referència:

In [51]:

```
# Creem un timedelta de 3 setmanes
d = datetime.timedelta(weeks=3)

# Calculem quin dia serà d'aquí a tres setmanes
future_3_weeks = now_1_dt + d
print("The date in three weeks will be: {}".format(future_3_weeks))

# Calculem quin dia era fa tres setmanes
past_3_weeks = now_1_dt - d
print("The date three weeks ago was: {}".format(past_3_weeks))
```

```
The date in three weeks will be: 2020-03-31 18:40:51.690380
The date three weeks ago was: 2020-02-18 18:40:51.690380
```

Es poden crear objectes `timedelta` (https://docs.python.org/3.8/library/datetime.html#timedelta-objects) especificant setmanes, dies, hores, minuts, segons, mil·lisegons o microsegons:

In [52]:

```
# Creem diversos timedeltas
one_day = datetime.timedelta(days=1)
eleven_seconds = datetime.timedelta(seconds=11)
five_hours = datetime.timedelta(hours=5)
eleven_and_so_seconds = datetime.timedelta(seconds=11.56845)
one_microsecond = datetime.timedelta(microseconds=1)

# Operem amb els timedeltas
print("One day in the future: {}".format(now_1_dt + one_day))
print("One day in the past: {}".format(now_1_dt - one_day))
print("Adding one microsecond: {}".format(now_1_dt + one_microsecond))
```

```
One day in the future: 2020-03-11 18:40:51.690380
One day in the past: 2020-03-09 18:40:51.690380
Adding one microsecond: 2020-03-10 18:40:51.690381
```

2.5.- El calendari

En els exemples que hem vist a l'apartat anterior, hem fet servir el mòdul `calendar` (https://docs.python.org/3.8/library/calendar.html#module-calendar) per a obtenir el nom del dia de la setmana corresponent a una data concreta. Les funcionalitats d'aquest mòdul, però, no es limiten al càlcul del dia de la setmana. El mòdul `calendar` (https://docs.python.org/3.8/library/calendar.html#module-calendar), com el seu nom indica, ens proveeix de funcionalitats relatives al calendari.

Un exemple de les funcionalitats més evidents d'un calendari és mostrar el propi calendari.

In [53]:

```
# Mostrem el calendari de l'any 2020
print(calendar.calendar(2020))
```

```
                                  2020

      de gener                     de febrer                    de març
dl dt dc dj dv ds dg      dl dt dc dj dv ds dg      dl dt dc dj dv d
s dg
       1  2  3  4  5                      1  2
1
 6  7  8  9 10 11 12       3  4  5  6  7  8  9       2  3  4  5  6
7  8
13 14 15 16 17 18 19      10 11 12 13 14 15 16       9 10 11 12 13 1
4 15
20 21 22 23 24 25 26      17 18 19 20 21 22 23      16 17 18 19 20 2
1 22
27 28 29 30 31            24 25 26 27 28 29         23 24 25 26 27 2
8 29
                                                    30 31

      d’abril                      de maig                      de juny
dl dt dc dj dv ds dg      dl dt dc dj dv ds dg      dl dt dc dj dv d
s dg
       1  2  3  4  5                   1  2  3       1  2  3  4  5
6  7
 6  7  8  9 10 11 12       4  5  6  7  8  9 10       8  9 10 11 12 1
3 14
13 14 15 16 17 18 19      11 12 13 14 15 16 17      15 16 17 18 19 2
0 21
20 21 22 23 24 25 26      18 19 20 21 22 23 24      22 23 24 25 26 2
7 28
27 28 29 30               25 26 27 28 29 30 31      29 30

     de juliol                    d’agost                      de setembre
dl dt dc dj dv ds dg      dl dt dc dj dv ds dg      dl dt dc dj dv d
s dg
       1  2  3  4  5                      1  2          1  2  3  4
5  6
 6  7  8  9 10 11 12       3  4  5  6  7  8  9       7  8  9 10 11 1
2 13
13 14 15 16 17 18 19      10 11 12 13 14 15 16      14 15 16 17 18 1
9 20
20 21 22 23 24 25 26      17 18 19 20 21 22 23      21 22 23 24 25 2
6 27
27 28 29 30 31            24 25 26 27 28 29 30      28 29 30
                          31

     d’octubre                    de novembre                  de desembre
dl dt dc dj dv ds dg      dl dt dc dj dv ds dg      dl dt dc dj dv d
s dg
          1  2  3  4                         1          1  2  3  4
5  6
 5  6  7  8  9 10 11       2  3  4  5  6  7  8       7  8  9 10 11 1
2 13
12 13 14 15 16 17 18       9 10 11 12 13 14 15      14 15 16 17 18 1
9 20
19 20 21 22 23 24 25      16 17 18 19 20 21 22      21 22 23 24 25 2
6 27
26 27 28 29 30 31         23 24 25 26 27 28 29      28 29 30 31
                          30
```

In [54]:

```
# Mostrem el calendari de l'octubre de 2020
print(calendar.month(2020, 10))
```

```
   d'octubre 2020
dl dt dc dj dv ds dg
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

Entre d'altres funcionalitats, el mòdul `calendar` (https://docs.python.org/3.8/library/calendar.html#module-calendar) ens permet saber si un any és o no de traspàs amb la funció `isleap` (https://docs.python.org/3.8/library/calendar.html#calendar.isleap), o bé comptar el número de dies de traspàs que hi ha entre dos anys:

In [55]:

```
# Calculem si l'any 2020 és de traspàs
calendar.isleap(2020)
```

Out[55]:

```
True
```

In [56]:

```
# Calculem el número de dies de traspàs entre els anys
# 2000 i 2050
calendar.leapdays(2020, 2050)
```

Out[56]:

```
8
```

2.6.- Recapitulant

A l'inici d'aquesta secció, ens plantejàvem un conjunt de preguntes relacionades amb la gestió de les dates i les hores que semblaven, a priori, poc directes de resoldre. Com a punt final de la secció, respondrem a aquestes preguntes fent servir el que hem après en aquesta unitat, i constatarem com de fàcil pot ser contestar-les amb les eines adequades.

En primer lloc, ens plantejàvem com obtenir el dia de la setmana (dilluns, dimarts, ..., diumenge) corresponent a una data concreta.

A partir d'un objecte `date` (https://docs.python.org/3.8/library/datetime.html#date-objects) (o `datetime` (https://docs.python.org/3.8/library/datetime.html#datetime-objects)) que representa una data, hem presentat dues maneres diferents d'obtenir el dia de la setmana:

In [57]:

```
# Creem un objecte date que representa una data concreta
a_date = datetime.date(year=2022, month=12, day=25)

# Alternativa 1: Obtenim el dia amb .weekday() i el nom del dia
# amb el mètode day_name del mòdul calendar
print('Day of Week:', calendar.day_name[a_date.weekday()])

# Alternativa 2: Fem servir strftime per obtenir el nom del dia de la setmana
print(a_date.strftime("Day of Week: %A"))
```

```
Day of Week: diumenge
Day of Week: diumenge
```

En segon lloc, volíem descobrir si les cadenes de caràcters "2019-11-15" i "Friday, November 15, 2019" corresponien o no a la mateixa data.

Podem crear dos objectes `date` (https://docs.python.org/3.8/library/datetime.html#date-objects) que representin les dates expressades per cadascuna de les cadenes de caràcters i, després, comparar-los per comprovar-ne la seva igualtat:

In [58]:

```
# Creem dos objectes date a partir de les cadenes de caràcters, especificant
# com a idioma l'anglès per a interpretar correctament la segona cadena
date_1 = datetime.datetime.strptime(
    "2019-11-15", "%Y-%m-%d")
locale.setlocale(locale.LC_ALL, 'en_GB.UTF-8')
date_2 = datetime.datetime.strptime(
    "Friday, November 15, 2019", "%A, %B %d, %Y")

# Comparem la igualtat de les dates creades
date_1 == date_2
```

Out[58]:

```
True
```

En tercer lloc, volíem saber quin dia serà d'aquí a 30 dies.

Podem combinar la funcionalitat d'obtenció de la data actual amb un `timedelta` (https://docs.python.org/3.8/library/datetime.html#timedelta-objects) que representi 30 dies per tal de saber quin dia serà d'aquí 30 dies:

In [59]:

```
# Creem un objecte amb la data actual
now_date = datetime.date.today()

# Creem un datetime de 30 dies
dt_30days = datetime.timedelta(days=30)

# Sumem els 30 dies a la data actual
print(now_date + dt_30days)
```

```
2020-04-09
```

Finalment, ens preguntàvem si l'any 2200 serà o no un any de traspàs.

El mòdul `calendar` (https://docs.python.org/3.8/library/calendar.html#module-calendar) ens permet comprovar si un any és de traspàs amb el mètode `isleap` (https://docs.python.org/3.8/library/calendar.html#calendar.isleap):

In [60]:

```
# Comprovem si l'any 2200 és un any de traspàs
calendar.isleap(2200)
```

Out[60]:

```
False
```

3.- Cadenes de caràcters

Ja coneixem algunes de les funcions de manipulació de cadenes de caràcters bàsiques, que ens permeten concatenar cadenes entre elles, formar cadenes a partir d'altres cadenes i de variables numèriques, particionar cadenes (fent servir *slicing*) i crear cadenes de caràcters a partir de llistes. Iniciarem aquesta secció fent un repàs d'aquestes funcions bàsiques:

In [61]:

```
# Creem 4 cadenes de caràcters
str_1 = "Today is:"
str_2 = "Friday"
str_3 = "Coding day"
pep_20 = "Beautiful is better than ugly.\nExplicit is better than implicit."
```

In [62]:

```
# Creem noves cadenes concatenant les cadenes anteriors
# amb espais que les separen
str_4 = str_1 + " " + str_2
print(str_4)
str_5 = str_1 + " " + str_3
print(str_5)
```

```
Today is: Friday
Today is: Coding day
```

In [63]:

```
# Creem cadenes a partir de valors de variables numèriques
str_6 = str_1 + " my {}th day of programming".format(6)
print(str_6)
```

```
Today is: my 6th day of programming
```

In [64]:

```
# També podem fer servir la funcionalitat de les cadenes-f
# per formatar text de manera compacta
days = 6
f'{str_1} my {days}th day of programming'
```

Out[64]:

```
'Today is: my 6th day of programming'
```

In [65]:

```
# Mostrem els 9 primers caràcters de la cadena pep_20
print(pep_20[:9])
# Mostrem els caràcters del 25 al 29 (aquest últim no inclòs)
print(pep_20[25:29])
# Mostrem l'últim caràcter
print(pep_20[-1:])
```

```
Beautiful
ugly
.
```

In [66]:

```
# Creem una cadena a partir d'una llista de cadenes
a_list_of_str = ["A", "B", "C", "D"]
"".join(a_list_of_str)
```

Out[66]:

```
'ABCD'
```

In [67]:

```
# Creem una cadena de caràcters amb els números del 0 al 9
# separats per espais
" ".join([str(x) for x in list(range(10))])
```

Out[67]:

```
'0 1 2 3 4 5 6 7 8 9'
```

In [68]:

```
# Creem una cadena de caràcters amb els números del 0 al 9
# separats per comes i espais
", ".join([str(x) for x in list(range(10))])
```

Out[68]:

```
'0, 1, 2, 3, 4, 5, 6, 7, 8, 9'
```

3.1- Manipulació de cadenes de caràcters

A l'hora de processar una cadena de caràcters, sovint serà necessari particionar-la, per tal de manipular-ne els fragments de manera individual. La funció `split` (https://docs.python.org/3.8/library/stdtypes.html#str.split) permet particionar una cadena de caràcters, obtenint com a resultat una llista de subcadenes de la cadena original.

Cridada sense arguments, la funció `split` (https://docs.python.org/3.8/library/stdtypes.html#str.split) fa servir caràcters en blanc (espais, tabuladors, salts de línia, etc.) com a separadors, i retorna una llista de les subcadenes separades, ometent aquests separadors:

In [69]:

```
# Separem la cadena pep_20
print(pep_20)
pep_20.split()
```

```
Beautiful is better than ugly.
Explicit is better than implicit.
```

Out[69]:

```
['Beautiful',
 'is',
 'better',
 'than',
 'ugly.',
 'Explicit',
 'is',
 'better',
 'than',
 'implicit.']
```

Fixeu-vos que les paraules 'ugly.' i 'Explicit' queden separades com a dos subcadenes diferents, ja que el salt de línia entre una i l'altre s'interpreta com a separador. En canvi, si volem que se separin les paraules considerant únicament els espais en blanc, podem indicar-ho afegint l'argument `sep` a la crida:

In [70]:

```
# Separem la cadena pep_20 considerant només els espais
# com a separadors
pep_20.split(sep=" ")
```

Out[70]:

```
['Beautiful',
 'is',
 'better',
 'than',
 'ugly.\nExplicit',
 'is',
 'better',
 'than',
 'implicit.']
```

Podem fer servir qualsevol caràcter (o cadena) com a separador:

In [71]:

```
# Separem la cadena per les lletres "a"
pep_20.split("a")
```

Out[71]:

```
['Be', 'utiful is better th', 'n ugly.\nExplicit is better th', 'n i
mplicit.']
```

In [72]:

```
# Separem la cadena fent servir la cadena "better" com a separador
pep_20.split("better")
```

Out[72]:

```
['Beautiful is ', ' than ugly.\nExplicit is ', ' than implicit.']
```

Una altra de les operacions de processament de cadenes d'ús comú és fer un recompte del número d'aparicions d'una subcadena dins d'una cadena:

In [73]:

```
# Comptem quantes vegades apareix la lletra "a" a la cadena pep_20
pep_20_a_num = pep_20.count("a")
print("PEP20 string has {} 'a'".format(pep_20_a_num))
# Comptem quantes vegades apareix la subcadena "better" a la cadena pep_20
pep_20_bet_num = pep_20.count("better")
print("PEP20 string has {} 'better'".format(pep_20_bet_num))
# Comptem quants espais en blanc té la cadena pep_20
pep_20_ws_num = pep_20.count(" ")
print("PEP20 string has {} whitespaces".format(pep_20_ws_num))
```

```
PEP20 string has 3 'a'
PEP20 string has 2 'better'
PEP20 string has 8 whitespaces
```

Hem vist com comptar el número d'aparicions d'una subcadena. Com ho faríem, però, si volguéssim saber on són aquestes aparicions? La funció `find` (https://docs.python.org/3.8/library/stdtypes.html#str.find) ens retorna l'índex on s'inicia una subcadena, o retorna `-1` si tal subcadena no existeix:

In [74]:

```
# Busquem l'índex de la primera aparició de 'beter'
str_to_search = "beter"
ind_better = pep_20.find(str_to_search)
print("'beter' starts at position {}".format(ind_better))

# Busquem l'índex de la primera aparició de 'better'
str_to_search = "better"
ind_better = pep_20.find(str_to_search)
print("'better' starts at position {}".format(ind_better))
```

```
'beter' starts at position -1
'better' starts at position 13
```

La cadena `better` apareix dues vegades dins de la cadena emmagatzemada a la variable `pep_20`. La funció `find` (https://docs.python.org/3.8/library/stdtypes.html#str.find) retorna l'índex de la primera aparició. Si volem trobar els índexs de la resta d'aparicions, caldrà indicar un segon paràmetre a la crida de `find` (https://docs.python.org/3.8/library/stdtypes.html#str.find), especificant la posició d'inici de la cerca:

In [75]:

```
# Busquem l'índex de la segona aparició de 'better'
ind2_better = pep_20.find(str_to_search, ind_better+1)
print("Next 'better' starts at position {}".format(ind2_better))
```

```
Next 'better' starts at position 43
```

In [76]:

```
# Busquem l'índex de la tercera aparició de 'better'
ind3_better = pep_20.find(str_to_search, ind2_better+1)
print("Next 'better' starts at position {}".format(ind3_better))
```

```
Next 'better' starts at position -1
```

Com que `better` només hi apareix dues vegades, la tercera vegada que fem la cerca la crida a `find` (https://docs.python.org/3.8/library/stdtypes.html#str.find) retorna `-1`, indicant que ja no hi ha més aparicions.

En algunes aplicacions, no estarem interessats en comptar el número d'aparicions d'una subcadena ni en saber on és aquesta subcadena, sinó simplement en substituir-la per alguna altra cadena. En aquests casos, podem fer servir `replace` (https://docs.python.org/3.8/library/stdtypes.html#str.replace), que reemplaça totes les aparicions d'una subcadena per una altra cadena:

In [77]:

```
# Reemplacem els espais en blanc per guions
print(pep_20.replace(" ", "_"), "\n")
# Reemplacem 'better' per 'worse'
print(pep_20.replace("better", "worse"), "\n")
# Eliminem els espais en blanc reemplaçant-los una cadena buida
print(pep_20.replace(" ", ""))
```

```
Beautiful_is_better_than_ugly.
Explicit_is_better_than_implicit.

Beautiful is worse than ugly.
Explicit is worse than implicit.

Beautifulisbetterthanugly.
Explicitisbetterthanimplicit.
```

A l'últim exemple s'eliminen tots els espais en blanc. A vegades, però, ens interessarà eliminar els espais en blanc que hi ha davant d'una cadena o darrere d'aquesta (o ambdós conjunts d'espais). Per a aquestes situacions, trobem les funcions `lstrip` (https://docs.python.org/3.8/library/stdtypes.html#str.lstrip) (eliminar els espais en blanc de l'esquerra o l'inici de la cadena), `rstrip` (https://docs.python.org/3.8/library/stdtypes.html#str.rstrip) (eliminar els espais de la dreta o el final de la cadena) i `strip` (https://docs.python.org/3.8/library/stdtypes.html#str.strip) (elimina els espais en blanc de davant i darrere de la cadena):

In [78]:

```
# Creem una cadena amb espais en blanc
str_with_spaces = "   A lot of   spaces "

# Mostrem el resultat d'aplicar rstrip, lstrip i strip a la cadena
print("String: #{}#\n".format(str_with_spaces))
print("rstrip: #{}#".format(str_with_spaces.rstrip()))
print("lstrip: #{}#".format(str_with_spaces.lstrip()))
print("strip:  #{}#".format(str_with_spaces.strip()))
```

```
String: #   A lot of   spaces #

rstrip: #   A lot of   spaces#
lstrip: #A lot of   spaces #
strip:  #A lot of   spaces#
```

És interessant notar que les funcions `strip` (https://docs.python.org/3.8/library/stdtypes.html#str.strip), `rstrip` (https://docs.python.org/3.8/library/stdtypes.html#str.rstrip) i `lstrip` (https://docs.python.org/3.8/library/stdtypes.html#str.lstrip) eliminen tots els espais en blanc que es troben en la posició que estan processant (davant, darrere o en les dues posicions) però, en canvi, no tenen cap efecte sobre la resta d'espais de la cadena (per exemple, l'espai en blanc entre 'A' i 'lot' mai es veu afectat per l'execució d'aquestes funcions).

3.2- Expressions regulars

Malgrat que les funcions que hem vist fins ara són molt potents, es queden curtes per afrontar molts dels problemes que trobarem a l'hora de preprocessar dades que continguin cadenes de text.

Per exemple, si volguéssim separar una cadena particionant-la per qualsevol vocal (en comptes de només la lletra 'a' com hem fet a l'exemple anterior), com ho faríem? Amb el que hem vist fins ara, hauríem d'anar separant vocal a vocal, processant en cada cas totes les subcadenes ja separades per les vocals anteriors.

Un altre exemple seria si volguéssim buscar no només la paraula 'better' dins d'una cadena, sinó qualsevol paraula de sis lletres, o potser buscar paraules que comencessin amb 'b' i acabessin amb 'tter', amb qualsevol lletra o lletres entre la 'b' i 'tter'. Com podríem implementar-ho amb les funcions que hem vist fins ara? Certament, seria possible, però el codi seria complex.

Per afrontar aquest tipus de problemes, s'acostumen a fer servir **expressions regulars** (també conegudes com a *regexes*, una contracció del terme en anglès, *regular expressions*). Una expressió regular és una expressió que fa servir un llenguatge especialitzat de descripció de cadenes de caràcters. Aquest llenguatge ens permet expressar amb facilitat condicions que descriuen com ha de ser una cadena, com ara les que hem exemplificat anteriorment (una cadena que representi qualsevol vocal, una cadena de sis lletres, una cadena que comenci amb 'b' i acabi amb 'tter', etc.). Aquestes expressions es poden fer servir, després, per tasques com les que hem exposat anteriorment: buscar subcadenes dins de cadenes, separar cadenes, substituir subcadenes per altres subcadenes, etc.

En aquest apartat, veurem doncs com podem fer servir expressions regulars per a processar cadenes de caràcters. En primer lloc, veurem les funcions que ens ofereix el mòdul `re` (https://docs.python.org/3.8/library/re.html). En segon lloc, presentarem una introducció a la sintaxi de les expressions regulars.

3.2.1- Funcions del mòdul `re`

El mòdul `re` (https://docs.python.org/3.8/library/re.html) és la interfície del motor d'expressions regulars per a Python. Per tant, caldrà importar-lo quan vulguem treballar amb *regexes*.

El mòdul té funcions equivalents a les que hem vist anteriorment, però que treballen amb expressions regulars en comptes de treballar amb cadenes de caràcters:

- `split` (https://docs.python.org/3.8/library/re.html#re.split): té el mateix comportament que el mètode `split` (https://docs.python.org/3.8/library/stdtypes.html#str.split) d' `str` (https://docs.python.org/3.8/library/stdtypes.html#str), permetent separar cadenes considerant un separador, que en aquest cas estarà expressat amb una expressió regular.
- `search` (https://docs.python.org/3.8/library/re.html#re.search): té un comportament similar a `find` (https://docs.python.org/3.8/library/stdtypes.html#str.find) d' `str` (https://docs.python.org/3.8/library/stdtypes.html#str), permetent trobar la primera aparició d'una subcadena dins d'una cadena. En aquest cas, la subcadena a buscar s'expressa com una expressió regular.
- `sub` (https://docs.python.org/3.8/library/re.html#re.sub): té un comportament similar a `replace` (https://docs.python.org/3.8/library/stdtypes.html#str.replace) d' `str` (https://docs.python.org/3.8/library/stdtypes.html#str), permetent reemplaçar subcadenes per una altra cadena. En aquest cas, la subcadena a reemplaçar s'expressa com una expressió regular.

Addicionalment, també incorpora altres funcions que són d'utilitat, com ara:

- `findall` (https://docs.python.org/3.8/library/re.html#re.findall): que permet buscar totes les aparicions (no solapades) d'una subcadena dins d'una altra cadena.

A continuació, veurem com podem fer servir aquestes funcions sense aprofitar la funcionalitat de les expressions regulars (que presentarem al següent apartat).

La funció `split` (https://docs.python.org/3.8/library/re.html#re.split) actua de manera anàloga a l' `split` (https://docs.python.org/3.8/library/stdtypes.html#str.split) d' `str` (https://docs.python.org/3.8/library/stdtypes.html#str), retornant una llista amb les subcadenes separades per

In [79]:

```
# Importem el mòdul re
import re

# Separem la cadena per les lletres "a"
re.split('a', pep_20)
```

Out[79]:

```
['Be', 'utiful is better th', 'n ugly.\nExplicit is better th', 'n i
mplicit.']
```

Ara, però, la primera cadena (`a`) podrà ser també una expressió regular, que ens descrigui com ha de ser el separador a fer servir.

La funció `search` (https://docs.python.org/3.8/library/re.html#re.search) busca la primera aparició d'una subcadena en una altra cadena. A diferència de `find` (https://docs.python.org/3.8/library/stdtypes.html#str.find), no retorna directament l'índex d'inici de la coincidència, sinó un objecte `match` (https://docs.python.org/3.8/library/re.html#match-objects) que conté informació addicional sobre la coincidència:

In [80]:

```
# Busquem l'índex de la primera aparició de 'better'
str_to_search = "better"
srch_better = re.search(str_to_search, pep_20)
print("The returned object is: {}".format(type(srch_better)))
print("that contains: {}".format(srch_better))
print("Substring starts at {} and ends at {}".format(
    srch_better.start(), srch_better.end()))
```

```
The returned object is: <class '_sre.SRE_Match'>
that contains: <_sre.SRE_Match object; span=(13, 19), match='bette
r'>
Substring starts at 13 and ends at 19
```

En canvi, la funció `findall` (https://docs.python.org/3.8/library/re.html#re.findall) ens retornarà una llista amb totes les coincidències d'una expressió regular:

In [81]:

```
# Busquem totes les aparicions de 'better'
srch_better = re.findall(str_to_search, pep_20)
print("The returned object is: {}".format(type(srch_better)))
print("that contains: {}".format(srch_better))
```

```
The returned object is: <class 'list'>
that contains: ['better', 'better']
```

Finalment, la funció `sub` (https://docs.python.org/3.8/library/re.html#re.sub) reemplaça totes les aparicions d'una subcadena per una altra:

In [82]:

```
# Reemplacem els espais en blanc per guions baixos
print(re.sub(" ", "_", pep_20))
```

```
Beautiful_is_better_than_ugly.
Explicit_is_better_than_implicit.
```

Fins ara hem vist com el mòdul `re` (https://docs.python.org/3.8/library/re.html) ens permet fer les mateixes operacions que havíem vist sobre cadenes de caràcters. Anem a veure ara com podem explotar el potencial d'aquest mòdul fent servir el llenguatge de les expressions regulars.

Les funcions que hem vist accepten un primer paràmetre que representa la subcadena a buscar, i que es fa servir com a separador (en `split` (https://docs.python.org/3.8/library/re.html#re.split)), com a objectiu de la cerca (`search` (https://docs.python.org/3.8/library/re.html#re.search) o `findall` (https://docs.python.org/3.8/library/re.html#re.findall)) o bé com a cadena a reemplaçar (`sub` (https://docs.python.org/3.8/library/re.html#re.sub)). Més enllà de ser una cadena de caràcters, aquest primer paràmetre pot fer servir el llenguatge de les expressions regulars per tal d'especificar condicions complexes sobre les cadenes a buscar.

3.2.2- Sintaxi de les expressions regulars

En una expressió regular distingim dos tipus de caràcters:

- Caràcters **normals**, que representen el propi caràcter. Per exemple, una 'a' és una expressió regular vàlida que representa una cadena de caràcters d'un sol caràcter que és la pròpia lletra 'a'.
- Caràcters **especials**, que serveixen o bé per representar grups de caràcters o bé per explicar com cal interpretar les expressions que els envolten.

Així, per exemple, alguns caràcters especials es fan servir per representar **grups de caràcters**:

- `.` : representa qualsevol caràcter (per defecte, qualsevol caràcter excepte la nova línia).
- `\d` : representa qualsevol dígit decimal.
- `\s` : representa qualsevol espai en blanc.
- `\w` : representa caràcters que poden formar part d'una paraula (lletres majúscules i minúscules, números, el guionet baix).
- `[]` : permet indicar un conjunt de caràcters, ja sigui individualment (`[abc]` permetria representar les cadenes `a`, `b` o `c`), o com a rang (`[A-Z]` representa totes les lletres en majúscula).

Altres caràcters especials donen informació sobre **com interpretar** les expressions:

- `|` : implementa l'operador OR, i permet crear una expressió regular que coincideixi o bé amb una expressió regular o bé amb una altra.

També hi ha un conjunt de caràcters especials que permeten especificar **repeticions** d'expressions regulars. Així, els caràcters següents indiquen les repeticions de l'expressió que les precedeix:

- `*` : 0 o més aparicions.
- `+` : 1 o més aparicions.
- `?` : 0 o 1 aparició.
- `{m}` : exactament *m* aparicions.
- `{m,n}` : entre *m* i *n* aparicions. Si un dels dos valors s'omet, aleshores només s'informa del mínim o del màxim d'aparicions.

3.2.3- Ús d'expressions regulars en la manipulació de cadenes

Ara que ja coneixem les funcions de manipulació de cadenes del mòdul `re` (https://docs.python.org/3.8/library/re.html) i la sintaxi bàsica del llenguatge de les expressions regulars, anem a veure com podem explotar el potencial d'ambdues funcionalitats juntes a través d'un seguit d'exemples. En primer lloc, veurem com podem resoldre alguns dels problemes que hem comentat anteriorment. En segon lloc, veurem exemples addicionals d'ús d'expressions regulars.

Si volem separar una cadena de caràcters utilitzant qualsevol vocal com a separador, podem fer servir una expressió regular que representi totes les vocals i utilitzar-la amb la funció `split` (https://docs.python.org/3.8/library/re.html#re.split). Els caràcters especials `[]` ens permeten expressar un conjunt de caràcters individuals, de manera que podem fer-los servir per expressar les vocals:

In [83]:

```
print(pep_20)

# Separem la cadena per qualsevol vocal
re.split(r'[aeiouAEIOU]', pep_20)
```

```
Beautiful is better than ugly.
Explicit is better than implicit.
```

Out[83]:

```
['B',
 '',
 '',
 't',
 'f',
 'l ',
 's b',
 'tt',
 'r th',
 'n ',
 'gly.\n',
 'xpl',
 'c',
 't ',
 's b',
 'tt',
 'r th',
 'n ',
 'mpl',
 'c',
 't.']
```

Fixeu-vos com la llista conté diverses cadenes buides, que resulten de separar vocals contigües. D'altra banda, és interessant notar també que fem servir el prefix 'r' per indicar la cadena de caràcters que conforma l'expressió regular. El prefix 'r' indica que la cadena s'ha d'interpretar com a literal *en brut* (en anglès, *raw string literal*).

Si volem separar les paraules, podem fer servir el caràcter especial `\s`, per tenir en compte qualsevol espai en blanc com a separador (tant l'espai `` ` `` com el salt de línia `\n` `s'utilitzarien com a separador per a la cadena pep_20`):

In [84]:

```
re.split(r'\s', pep_20)
```

Out[84]:

```
['Beautiful',
 'is',
 'better',
 'than',
 'ugly.',
 'Explicit',
 'is',
 'better',
 'than',
 'implicit.']
```

En canvi, si volem separar les frases, podríem fer servir com a separador un el punt seguit d'un espai en blanc. Fixeu-vos que incloem l'espai en blanc després del punt, ja que sinó l'última frase quedaria separada en dos pel punt final (la frase en sí i una cadena buida), i el salt de línia quedaria inclòs a la segona frase:

In [85]:

```
# Separem la cadena per frases: les frases se separen per un punt
# El resultat inclou una cadena buida al final, ja que l'últim punt
# s'utilitza com a separador, així com el salt de línia
re.split(r'\.', pep_20)
```

Out[85]:

```
['Beautiful is better than ugly', '\nExplicit is better than implici
t', '']
```

In [86]:

```
# Separem la cadena per frases: les frases se separen per un punt seguit
# d'un espai en blanc
re.split(r'\.\s', pep_20)
```

Out[86]:

```
['Beautiful is better than ugly', 'Explicit is better than implici
t.']
```

És important notar que per representar el punt, hem fet servir l'expressió `\.` en comptes de `.`. El motiu és que, com hem vist, `.` és un caràcter especial que coincideix amb qualsevol caràcter. Si, pel contrari, volem que l'expressió regular només coincideixi quan inclogui un punt (el caràcter '.'), aleshores caldrà **escapar** la seqüència, fent servir la contrabarra (`\`).

Un altre exemple que ens plantejàvem era com buscar paraules que comencin amb 'b' i acabin amb 'tter', amb qualsevol lletra o lletres entremig. Anem a veure com podríem enfocar aquest exemple. En primer lloc, veurem com buscar paraules que comencin amb 'b' i acabin amb 'tter', amb una única lletra entremig:

In [87]:

```
# Creem una variant de la cadena pep_20, que ens permetrà exemplificar
# les diferents casuístiques
pep_20_bis = "Beautiful is better than ugly.\nExplicit is better than " \
    "implicit. Butter Much Webetter. bitter. bYYYtter or btter. python."

# Mostrem les cadenes d'exemple
print(pep_20)
print("\n")
print(pep_20_bis)
```

```
Beautiful is better than ugly.
Explicit is better than implicit.


Beautiful is better than ugly.
Explicit is better than implicit. Butter Much Webetter. bitter. bYYY
tter or btter. python.
```

Una primera aproximació seria definir una expressió regular que comenci amb 'b', segueixi amb qualsevol caràcter que pugui formar part d'una paraula (que podem expressar amb el caràcter especial `\w`), i acabi amb 'tter':

In [88]:

```
print(r"b\wtter in pep_20: {}".format(re.findall(r"b\wtter", pep_20)))
print(r"b\wtter in pep_20_bis: {}".format(re.findall(r"b\wtter", pep_20_bis)))
```

```
b\wtter in pep_20: ['better', 'better']
b\wtter in pep_20_bis: ['better', 'better', 'better', 'bitter']
```

L'expressió sembla retornar els resultats esperats per a la cadena `pep_20` però, en canvi, per a la cadena `pep_20_bis` ens retorna un 'better' addicional i no és capaç d'identificar 'Butter'. Els motius d'aquest comportament són, d'una banda, que per defecte les expressions regulars són *case sensitive*, de manera que 'B' i 'b' es consideren cadenes diferents (i, per tant, 'Butter' no és una coincidència vàlida); i, d'altra banda, que no estem especificant delimitadors de les paraules, de manera que 'Webetter' és una coincidència vàlida per 'better'.

Si volem que s'ignorin si les lletres són majúscules o minúscules a l'hora d'avaluar una expressió regular, podem fer servir el flag `IGNORECASE` a la crida:

In [89]:

```
print(r"b\wtter in pep_20_bis: {}".format(
    re.findall(r"b\wtter", pep_20_bis, re.IGNORECASE)))
```

```
b\wtter in pep_20_bis: ['better', 'better', 'Butter', 'better', 'bit
ter']
```

D'aquesta manera, 'Butter' s'identifica correctament.

D'altra banda, si volem buscar paraules completes (que no formin part d'altres paraules) podem especificar el(s) caràcter(s) que esperem trobar davant i darrere de la paraula:

In [90]:

```
# Delimitem les paraules per espais en blanc
print(r" b\wtter  in pep_20_bis: {}".format(
    re.findall(r" b\wtter ", pep_20_bis, re.IGNORECASE)))
# Delimitem les paraules per espais en blanc o bé punts
print(r"[ \.]b\wtter[ \.] in pep_20_bis: {}".format(
    re.findall(r"[ \.]b\wtter[ \.]", pep_20_bis, re.IGNORECASE)))
```

```
 b\wtter  in pep_20_bis: [' better ', ' better ', ' Butter ']
[ \.]b\wtter[ \.] in pep_20_bis: [' better ', ' better ', ' Butter
', ' bitter.']
```

En segon lloc, veurem ara com buscar paraules que comencin amb 'b' i acabin amb 'tter', però acceptant més d'una lletra entremig. A partir de l'expressió que hem fet servir per a una sola lletra entre les dues subcadenes, podem indicar que acceptem qualsevol número de repeticions de la lletra entre 'b' i 'tter' amb el modificador `*`, o bé exigir que, com a mínim, n'hi hagi una amb `+`:

In [91]:

```
print("Using *: {}".format(
    re.findall(r"[ \.]b\w*tter[ \.]", pep_20_bis, re.IGNORECASE)))
print("Using +: {}".format(
    re.findall(r"[ \.]b\w+tter[ \.]", pep_20_bis, re.IGNORECASE)))
```

```
Using *: [' better ', ' better ', ' Butter ', ' bitter.', ' bYYYtter
', ' btter.']
Using +: [' better ', ' better ', ' Butter ', ' bitter.', ' bYYYtter
']
```

Per últim, podríem fer servir una estratègia similar per identificar totes les paraules de sis lletres, delimitant-les per espais o bé punts, i fent servir el caràcter especial `\w` repetit exactament sis vegades:

In [92]:

```
# Repetint el caràcter \w 6 vegades explícitament:
print("6 letter words: {}".format(
    re.findall(r"[ \.]\w\w\w\w\w\w[ \.]", pep_20_bis)))
# Especificant el número de repeticions amb {}:
print("6 letter words: {}".format(
    re.findall(r"[ \.]\w{6}[ \.]", pep_20_bis)))
```

```
6 letter words: [' better ', ' better ', ' Butter ', ' bitter.', ' p
ython.']
6 letter words: [' better ', ' better ', ' Butter ', ' bitter.', ' p
ython.']
```

Aquesta secció ha presentat una petita introducció a l'ús d'expressions regulars amb el mòdul `re` (https://docs.python.org/3.8/library/re.html) de Python. Ara bé, hi ha molts altres caràcters especials i construccions que es poden fer amb el llenguatge de les expressions regulars, i que ens poden ser útils per a processar cadenes de caràcters. Per a continuar l'aprenentatge de l'ús d'expressions regulars en Python, us recomanem llegir el *Regular Expression HOWTO* (https://docs.python.org/3/howto/regex.html#regex-howto) de la documentació oficial de Python.

Addicionalment, si voleu descobrir tots els detalls sobre el llenguatge d'expressions regulars i sobre les funcions que implementa el mòdul `re` de Python, podeu dirigir-vos a la pàgina oficial del mòdul `re` (https://docs.python.org/3/library/re.html).

4.- Exercicis per practicar

A continuació hi trobareu un conjunt de problemes que us poden servir per a practicar els conceptes explicats en aquesta unitat. Us recomanem que intenteu fer aquests problemes vosaltres mateixos i que, una vegada realitzats, compareu la solució que us proposem amb la vostra solució. No dubteu en adreçar tots els dubtes que sorgeixin de la resolució d'aquests exercicis o bé de les solucions proposades al fòrum de l'aula.

1. Decidiu quina és l'estructura de dades més adequada per respondre a cadascuna de les preguntes següents i escriviu el codi que permeti respondre-les.

Disposem de dades d'un conjunt de ciutats del món. De cada ciutat, en sabem si té més de 14 milions d'habitants, si és capital del país on es troba, i si té una densitat de població per sobre els 20 000 habitants per km^2:

- Les ciutats Shanghai, Beijing, Delhi, Istanbul, Karachi, Guangzhou i Kinshasa tenen més de 14 milions d'habitants. La resta de ciutats de les quals en tenim dades no tenen més de 14 milions d'habitants.
- Les ciutats Delhi, Beijing, Kinshasa, Tokyo, Moscow, Jakarta, Seoul i Cairo són capitals del país on es troben.La resta de ciutats de les quals en tenim dades no són capitals.
- Les ciutats Cairo, Kinshasa, Delhi i Tokyo tenen una densitat de població per sobre els 20 000 habitants per km^2. La resta de ciutats de les quals en tenim dades no superen els 20 000 habitants per km^2.

1.1. De quantes ciutats (diferents) en tenim dades? Assumirem que no hi ha cap ciutat que no compleixi com a mínim una de les propietats anteriors.

In [93]:

```
# Resposta
```

1.2. Quantes ciutats tenen més de 14 milions d'habitants i una densitat de població per sobre dels 20 000 habitants per km^2?

In [94]:

```
# Resposta
```

1.3. Quines ciutats tenen una densitat de població per sobre dels 20 000 habitants per km^2 però no més de 14 milions d'habitants?

In [95]:

```
# Resposta
```

1.4. Quin és el país amb major número de ciutats per sobre de 14 milions d'habitants? Quantes ciutats d'aquestes característiques hi ha a cada país?

Per respondre aquestes preguntes, ens faltarà afegir informació al conjunt de dades de ciutats del qual disposem per fer l'activitat. Penseu quina seria l'estructura de dades més adient per a emmagatzemar aquesta informació extra i calculeu la resposta a la pregunta plantejada.

In [96]:

```
# Resposta
```

1.5. Quins són els dos països adjacents que tenen el major número de ciutats per sobre de 14 milions d'habitants?

Per respondre aquesta pregunta, ens faltarà afegir més informació al conjunt de dades de ciutats del qual disposem per fer l'activitat. Penseu quina seria l'estructura de dades més adient per a emmagatzemar aquesta informació extra i calculeu la resposta a la pregunta plantejada.

In [97]:

```
# Resposta
```

1. Calculeu quantes hores ha treballat la persona que ha escrit la següent frase:

   "I started working at 17:22:42 and finished at 22:00:00"

In [98]:

```
sentence = "I started working at 17:22:42 and finished at 22:00:00"

# Resposta
```

1. Donada la cadena de caràcters `sentence`, reemplaceu tots els espais en blanc per punts.

In [99]:

```
sentence = " From     time to time, Python makes  an    incompatible change " \
    " to the   advertised semantics of core language constructs   "

# Resposta
```

1. Donada la mateixa cadena de caràcters de l'exercici anterior, reemplaceu tots els espais en blanc contigus per un únic punt. És a dir, si trobeu tres espais en blanc consecutius, aquests s'han de reemplaçar per un únic punt, i no per tres punts com implementàvem a l'exercici anterior. Elimineu els espais que es troben a l'inici i al final de la cadena abans de substituir-los per punts.

In [100]:

```
# Resposta
```

1. Proporcioneu una llista amb totes les paraules de quatre lletres de la cadena de caràcters `sentence` que comencin per `t o F`.

In [101]:

```
# Resposta
```

1. Reemplaceu totes les majúscules de la cadena `sentence` per interrogants.

In [102]:

```
# Resposta
```

1. Una aerolínia ens contracta per ajudar-la a optimitzar el procediment d'embarcament del seus avions. L'aerolínia disposa de tres classes de bitllets, primera, *business* i turista. A l'hora d'embarcar els clients se situen en tres cues, una per cada classe. Després, però, només hi ha dues hostesses que els validin la targeta d'embarcament, de manera que les tres cues inicials es converteixen en dues cues, a partir de les quals els passatgers accedeixen als avions.

Actualment, l'aerolínia fa servir la següent estratègia per convertir les tres cues inicials (per classe) en les dues cues que embarquen (una per cada hostessa):

D'una banda, els clients de primera i *business* van a parar a la cua 1 (cua prioritària), intercalant un client de cada tipus a la cua prioritària sempre que hi hagi prou clients per fer-ho, i assignant després tots els clients restants a la nova cua. Així, el primer client que s'assigna a la cua prioritària és el primer client de la classe primera, el segon client de la cua prioritària serà el primer client de la cua de *business*, el tercer client de la cua prioritària serà el segon client de la classe primera, etc.

D'altra banda, els clients de classe turista van a parar a la cua 2 (cua no prioritària), seguint l'ordre que tenien a la cua de la classe turista. Ara bé, si un client de classe turista té mobilitat reduïda o va acompanyat de nens, aleshores aquest se situa al capdavant de la cua prioritària. Si hi ha més d'un client en aquestes condicions, l'ordre que segueixen a la cua de la classe turista es manté.

Per tal d'avaluar com de bona és l'estratègia, l'aerolínia fa servir dues mètriques:

1. El temps que es tarda en embarcar l'avió és 30 segons per passatger, considerant que les dues cues (1 i 2) embarquen alhora. És a dir, si la cua 1 té un passatger i la cua 2 en té dos, es tardarà un minut en embarcar.
2. La satisfacció global dels seus clients, que es calcula fent la mitjana de la satisfacció de cada client, considerant que:

- Els clients de primera tenen una satisfacció de -25 vegades el número de posicions que han perdut a la cua prioritària respecte a la seva posició original a la cua de primera. És a dir, un client que estava en 3a posició a la cua de primera i que ocupa la 5ena posició de la cua prioritària, tindrà una satisfacció de -50.
- Els clients de *business* sempre tenen una satisfacció de 0.
- Els clients amb bitllet de classe turista que han estat moguts a la cua prioritària tenen una satisfacció de 100. En canvi, els que han estat moguts a la cua no prioritària tenen una satisfacció de 25 si han avançat alguna posició a la cua 2 respecte a la seva posició inicial a la cua de la classe turista, o de 0 en qualsevol altre cas.

Com a analistes de dades, avaluarem la satisfacció i el temps de pujada a l'avió del vol 714.

7.1 Carregueu les dades dels passatgers del vol 714 que trobareu al dataset `data/flight714.csv` i creeu les tres cues primera, *business* i turista amb les dades dels passatgers. Els clients es troben ordenats segons la seva posició a la cua, amb la columna `client_class` indicant a quina de les cues pertanyen.

In [103]:

```
# Resposta
```

7.2 Mostreu quants passatgers hi ha a cada cua, amb el detall de quants d'ells tenen o bé criatures o bé mobilitat reduïda.

In [104]:

```
# Resposta
```

7.3 Implementeu una funció que generi les dues cues d'embarcament (cua prioritària i cua no prioritària) a partir de les tres cues obtingudes segons la classe del bitllet del passatger.

In [105]:

```
# Resposta
```

7.4 Implementeu una funció que calculi el temps que es tarda en embarcar l'avió i una funció que calculi la satisfacció dels passatgers.

In [106]:

```
# Resposta
```

7.5 Calculeu el temps que es tarda en embarcar el vol 714 i la satisfacció dels clients.

In [107]:

```
# Resposta
```

4.1.- Solucions als exercicis per practicar

1. Decidiu quina és l'estructura de dades més adequada per respondre a cadascuna de les preguntes següents i escriviu el codi que permeti respondre-la.

Disposem de dades d'un conjunt de ciutats del món. De cada ciutat, en sabem si té més de 14 milions d'habitants, si és capital del país on es troba, i si té una densitat de població per sobre els 20 000 habitants per km^2:

- Les ciutats Shanghai, Beijing, Delhi, Istanbul, Karachi, Guangzhou i Kinshasa tenen més de 14 milions d'habitants. La resta de ciutats de les quals en tenim dades no tenen més de 14 milions d'habitants.
- Les ciutats Delhi, Beijing, Kinshasa, Tokyo, Moscow, Jakarta, Seoul i Cairo són capitals del país on es troben.La resta de ciutats de les quals en tenim dades no són capitals.
- Les ciutats Cairo, Kinshasa, Delhi i Tokyo tenen una densitat de població per sobre els 20 000 habitants per km^2. La resta de ciutats de les quals en tenim dades no superen els 20 000 habitants per km^2.

1.1. De quantes ciutats (diferents) en tenim dades? Assumirem que no hi ha cap ciutat que no compleixi com a mínim una de les propietats anteriors.

In [108]:

```
# Resposta

# Podem desar les dades en 3 conjunts, un per cadascuna de les propietats
# que volem analitzar i calcular la unió d'aquests conjunts

more_than_14M = {"Shanghai", "Beijing", "Delhi", "Istanbul", "Karachi",
                 "Guangzhou", "Kinshasa"}
capital = {"Delhi", "Beijing", "Kinshasa", "Tokyo", "Moscow", "Jakarta",
           "Seoul", "Cairo"}
more_than_20K = {"Cairo", "Kinshasa", "Delhi", "Tokyo"}

print("We have data from {} different cities".format(
    len(more_than_14M.union(capital).union(more_than_20K))))
```

```
We have data from 12 different cities
```

1.2. Quantes ciutats tenen més de 14 milions d'habitants i una densitat de població per sobre dels 20 000 habitants per km^2?

In [109]:

```
# Resposta

# Podem aprofitar els 2 conjunts d'interès (more_than_14M i more_than_20K)
# creats a l'apartat anterior, i calcular-ne la intersecció.

print("There are {} cities with more than 14M inhabitants and a density "
      "over 20K/km^2".format(len(more_than_14M.intersection(more_than_20K))))
```

```
There are 2 cities with more than 14M inhabitants and a density over
20K/km^2
```

1.3. Quines ciutats tenen una densitat de població per sobre dels 20 000 habitants per km^2 però no més de 14 milions d'habitants?

In [110]:

```
# Resposta

# De nou, podem aprofitar els 2 conjunts d'interès (more_than_14M i
# more_than_20K) creats al primer apartat, i calcular-ne ara la diferència.

print("Cities with a density over 20K/km^2 and less than 14M inhabitants "
      "different cities: {}".format(more_than_20K.difference(more_than_14M)))
```

```
Cities with a density over 20K/km^2 and less than 14M inhabitants di
fferent cities: {'Tokyo', 'Cairo'}
```

1.4. Quin és el país amb major número de ciutats per sobre de 14 milions d'habitants? Quantes ciutats d'aquestes característiques hi ha a cada país?

Per respondre aquestes preguntes, ens faltarà afegir informació al conjunt de dades de ciutats del qual disposem per fer l'activitat. Penseu quina seria l'estructura de dades més adient per a emmagatzemar aquesta informació extra i calculeu la resposta a la pregunta plantejada.

Resposta:

In [111]:

```
# Resposta

# Desem el país al qual pertany cadascuna de les ciutats amb més
# de 14M d'habitants
city_country = {'Shanghai': 'China', 'Beijing': 'China',
                'Delhi': 'India', 'Istanbul': 'Turkey',
                'Karachi': 'Pakistan', 'Guangzhou': 'China',
                'Kinshasa': 'Congo'}

# Comptem quantes ciutats de més de 14M d'habitants hi ha a cada país
cities_per_country = defaultdict(get_0)
for k, v in city_country.items():
    cities_per_country[v] += 1

print(cities_per_country)

# Recuperem el país amb més ciutats
# Opció 1: recorrent el diccionari amb un bucle
max_val = -1
city = None
for k, v in cities_per_country.items():
    if v > max_val:
        max_val = v
        city = k
print(city)
```

```
defaultdict(<function get_0 at 0x7f2e74565598>, {'China': 3, 'Indi
a': 1, 'Turkey': 1, 'Pakistan': 1, 'Congo': 1})
China
```

In [112]:

```
# Opció 2: recuperem la clau del diccionari amb valor màxim
import operator
print(max(cities_per_country.items(), key=operator.itemgetter(1))[0])
```

```
China
```

1.5. Quins són els dos països adjacents que tenen el major número de ciutats per sobre de 14 milions d'habitants?

Per respondre aquesta pregunta, ens faltarà afegir més informació al conjunt de dades de ciutats del qual disposem per fer l'activitat. Penseu quina seria l'estructura de dades més adient per a emmagatzemar aquesta informació extra i calculeu la resposta a la pregunta plantejada.

In [113]:

```
# Resposta

cities_per_country.keys()
```

Out[113]:

```
dict_keys(['China', 'India', 'Turkey', 'Pakistan', 'Congo'])
```

In [114]:

```
# Desem les fronteres de cada País
# (ometem els països que no estan al dataset )
country_frontier = {
    'China': ['India', 'Pakistan', 'Russia'],
    'India': ['China', 'Pakistan', 'Indonesia'],
    'Turkey': [],
    'Pakistan': ['China', 'India'],
    'Congo': []}

# Calculem quins són els dos països adjacents amb més ciutats
# de més de 14M
cities_per_adj_countries = {}
countries = list(cities_per_country.keys())
for i in range(len(countries)):
    for j in range(i+1, len(countries)):
        country_1, country_2 = countries[i], countries[j]
        if country_1 in country_frontier[country_2]:
            cities_per_adj_countries[country_1 + '-' + country_2] = \
                cities_per_country[country_1] + cities_per_country[country_2]

cities_per_adj_countries
```

Out[114]:

```
{'China-India': 4, 'China-Pakistan': 4, 'India-Pakistan': 2}
```

1. Calculeu quantes hores ha treballat la persona que ha escrit la següent frase:

   "I started working at 17:22:42 and finished at 22:00:00"

In [115]:

```
sentence = "I started working at 17:22:42 and finished at 22:00:00"

# Resposta

start = datetime.datetime.strptime(sentence[21:29], "%H:%M:%S")
end = datetime.datetime.strptime(sentence[-8:], "%H:%M:%S")
working_time = end - start

print("The person has worked {} hours".format(working_time))
```

```
The person has worked 4:37:18 hours
```

1. Donada la cadena de caràcters `sentence` , reemplaceu tots els espais en blanc per punts.

In [116]:

```
sentence = " From     time to time, Python makes  an    incompatible change " \
    " to the   advertised semantics of core language constructs   "

# Resposta

sentence.replace(" ", ".")
```

Out[116]:

```
'.From.....time.to.time,.Python.makes..an....incompatible.change..t
o.the...advertised.semantics.of.core.language.constructs...'
```

1. Donada la mateixa cadena de caràcters de l'exercici anterior, reemplaceu tots els espais en blanc contigus per un únic punt. És a dir, si trobeu tres espais en blanc consecutius, aquests s'han de reemplaçar per un únic punt, i no per tres punts com implementàvem a l'exercici anterior. Elimineu els espais que es troben a l'inici i al final de la cadena abans de substituir-los per punts.

In [117]:

```
# Resposta

# Eliminem els espais en blanc
stripped_sentence = sentence.strip()
# Substituïm els espais en blanc per punts
re.sub(" +", ".", stripped_sentence)
```

Out[117]:

```
'From.time.to.time,.Python.makes.an.incompatible.change.to.the.adver
tised.semantics.of.core.language.constructs'
```

1. Proporcioneu una llista amb totes les paraules de quatre lletres de la cadena de caràcters `sentence` que comencin per `t` o `F`.

In [118]:

```
# Resposta

re.findall(" [t|F]\w{3} ", sentence)
```

Out[118]:

```
[' From ', ' time ']
```

```
3:19: W605 invalid escape sequence '\w'
```

1. Reemplaceu totes les majúscules de la cadena `sentence` per interrogants.

In [119]:

```
# Resposta

re.sub("[A-Z]", "?", sentence)
```

Out[119]:

```
' ?rom     time to time, ?ython makes  an    incompatible change  to
the   advertised semantics of core language constructs   '
```

1. Una aerolínia ens contracta per ajudar-la a optimitzar el procediment d'embarcament del seus avions. L'aerolínia disposa de tres classes de bitllets, primera, *business* i turista. A l'hora d'embarcar els clients se situen en tres cues, una per cada classe. Després, però, només hi ha dues hostesses que els validin la targeta d'embarcament, de manera que les tres cues inicials es converteixen en dues cues, a partir de les quals els passatgers accedeixen als avions.

Actualment, l'aerolínia fa servir la següent estratègia per convertir les tres cues inicials (per classe) en les dues cues que embarquen (una per cada hostessa):

D'una banda, els clients de primera i *business* van a parar a la cua 1 (cua prioritària), intercalant un client de cada tipus a la cua prioritària sempre que hi hagi prou clients per fer-ho, i assignant després tots els clients restants a la nova cua. Així, el primer client que s'assigna a la cua prioritària és el primer client de la classe primera, el segon client de la cua prioritària serà el primer client de la cua de *business*, el tercer client de la cua prioritària serà el segon client de la classe primera, etc.

D'altra banda, els clients de classe turista van a parar a la cua 2 (cua no prioritària), seguint l'ordre que tenien a la cua de la classe turista. Ara bé, si un client de classe turista té mobilitat reduïda o va acompanyat de nens, aleshores aquest se situa al capdavant de la cua prioritària. Si hi ha més d'un client en aquestes condicions, l'ordre que segueixen a la cua de la classe turista es manté.

Per tal d'avaluar com de bona és l'estratègia, l'aerolínia fa servir dues mètriques:

1. El temps que es tarda en embarcar l'avió és 30 segons per passatger, considerant que les dues cues (1 i 2) embarquen alhora. És a dir, si la cua 1 té un passatger i la cua 2 en té dos, es tardarà un minut en embarcar.
2. La satisfacció global dels seus clients, que es calcula fent la mitjana de la satisfacció de cada client, considerant que:

- Els clients de primera tenen una satisfacció de -25 vegades el número de posicions que han perdut a la cua prioritària respecte a la seva posició original a la cua de primera. És a dir, un client que estava en 3a posició a la cua de primera i que ocupa la 5ena posició de la cua prioritària, tindrà una satisfacció de -50.
- Els clients de *business* sempre tenen una satisfacció de 0.
- Els clients amb bitllet de classe turista que han estat moguts a la cua prioritària tenen una satisfacció de 100. En canvi, els que han estat moguts a la cua no prioritària tenen una satisfacció de 25 si han avançat alguna posició a la cua 2 respecte a la seva posició inicial a la cua de la classe turista, o de 0 en qualsevol altre cas.

Com a analistes de dades, avaluarem la satisfacció i el temps de pujada a l'avió del vol 714.

7.1 Carregueu les dades dels passatgers del vol 714 que trobareu al dataset `data/flight714.csv` i creeu les tres cues primera, *business* i turista amb les dades dels passatgers. Els clients es troben ordenats segons la seva posició a la cua, amb la columna `client_class` indicant a quina de les cues pertanyen.

In [120]:

```
# Resposta

import pandas as pd

df = pd.read_csv("data/flight714.csv")

bq, pq, tq = [], [], []
col_names = list(df.columns)

for i in range(len(df)):
    c = {c: df.iloc[i][c] for c in col_names}
    if c['client_class'] == "p":
        pq.append(c)
    if c['client_class'] == "b":
        bq.append(c)
    if c['client_class'] == "t":
        tq.append(c)
```

7.2 Mostreu quants passatgers hi ha a cada cua, amb el detall de quants d'ells tenen o bé criatures o bé mobilitat reduïda.

In [121]:

```
# Resposta

print("Passengers in first class: {}".format(len(pq)))
print("Passengers in business class: {}".format(len(bq)))
print("Passengers in tourist class: {}".format(len(tq)))

f_with_crm = sum([1 for p in pq
                  if p['has_children'] or p['has_reduced_mobility']])
b_with_crm = sum([1 for p in bq
                  if p['has_children'] or p['has_reduced_mobility']])
t_with_crm = sum([1 for p in tq
                  if p['has_children'] or p['has_reduced_mobility']])

print("Passengers in first class with children or reduced mob.: {}".
      format(f_with_crm))
print("Passengers in business class with children or reduced mob.: {}".
      format(b_with_crm))
print("Passengers in tourist class with children or reduced mob.: {}".
      format(t_with_crm))
```

```
Passengers in first class: 3
Passengers in business class: 10
Passengers in tourist class: 37
Passengers in first class with children or reduced mob.: 0
Passengers in business class with children or reduced mob.: 0
Passengers in tourist class with children or reduced mob.: 6
```

7.3 Implementeu una funció que generi les dues cues d'embarcament (cua prioritària i cua no prioritària) a partir de les tres cues obtingudes segons la classe del bitllet del passatger.

In [122]:

```
# Resposta

from copy import copy


def compute_new_queues_alg_1(in_bq, in_pq, in_tq):
    # Copiem les dades de les cues d'entrada per no modificar els
    # valors de les variables d'entrada
    bq, pq, tq = copy(in_bq), copy(in_pq), copy(in_tq)
    # Inicialitzem les dues cues de sortida buides
    q1, q2 = [], []

    # Movem als passatgers de la classe turista
    while len(tq) != 0:
        client = tq.pop(0)
        if client["has_children"] or client["has_reduced_mobility"]:
            q1.append(client)
        else:
            q2.append(client)

    # Movem a la resta de passatgers, alternant entre primera i bussiness
    while len(bq) != 0 or len(pq) != 0:
        if len(pq) != 0:
            client = pq.pop(0)
            q1.append(client)
        if len(bq) != 0:
            client = bq.pop(0)
            q1.append(client)

    return q1, q2
```

7.4 Implementeu una funció que calculi el temps que es tarda en embarcar l'avió i una funció que calculi la satisfacció dels passatgers.

In [123]:

```
# Resposta


def evaluate_time(q1, q2):
    time_per_pas = 30
    return max(len(q1), len(q2))*time_per_pas


def evaluate_sat(q1, q2, bq, pq, tq):

    num_pass = len(q1) + len(q2)

    # Tourists with children or reduced mobility
    t = sum([100 for p in q1 if p in tq])

    # Other tourists
    t2 = sum([25 for p in q2 if q2.index(p) < tq.index(p)])

    # First class
    p = sum([-50*(q1.index(p)-pq.index(p)) for p in q1
             if p in pq if q1.index(p) > pq.index(p)])

    return (t + t2 + p)/num_pass
```

7.5 Calculeu el temps que es tarda en embarcar el vol 714 i la satisfacció dels clients.

In [124]:

```
# Resposta

q1, q2 = compute_new_queues_alg_1(bq, pq, tq)
print("Time: {}".format(evaluate_time(q1, q2)))
print("Satisfaction: {}".format(evaluate_sat(q1, q2, bq, pq, tq)))
```

```
Time: 930
Satisfaction: 6.0
```

5.- Bibliografia

5.1.- Bibliografia bàsica

Us recomanem revisar la documentació oficial de les funcions i classes descrites en aquesta unitat, que trobareu enllaçades en cada un dels apartats que les descriuen.

A més, us recomanem revisar l'especificació completa del format d' `strftime i strptime` del mòdul `datetime` a la documentació oficial del mòdul (https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes). No cal que conegueu tots els comodins que s'usen per descriure dates i hores, però una lectura de la documentació us proporcionarà una visió global de quin tipus d'expressions es poden construir.

De manera anàloga, us recomanem llegir el *Regular Expression HOWTO* (https://docs.python.org/3/howto/regex.html#regex-howto) de la documentació oficial de Python.

5.2.- Bibliografia addicional - Ampliació de coneixements

En aquesta unitat hem presentat una petita introducció a l'ús d'expressions regulars amb el mòdul `re` (https://docs.python.org/3.8/library/re.html) de Python. Si voleu descobrir tots els detalls sobre el llenguatge d'expressions regulars i sobre les funcions que implementa el mòdul `re` de Python, podeu dirigir-vos a la pàgina oficial del mòdul `re` (https://docs.python.org/3/library/re.html).

D'altra banda, els exercicis resolts de la unitat fan ús de la funció `copy` . Si voleu llegir més informació sobre aquesta funció i per què és necessari el seu ús en l'exemple, us recomanem llegir la documentació oficial del mòdul `copy` (https://docs.python.org/3.8/library/copy.html) i aquesta pregunta (https://stackoverflow.com/questions/17246693/what-is-the-difference-between-shallow-copy-deepcopy-and-normal-assignment-oper) d'stackoverflow. Intenteu reflexionar sobre per què, en el cas de l'exemple, hem utilitzat `copy` en comptes de `deepcopy` i si el codi funcionaria correctament usant `deepcopy`