VALIDACIONS

COMPROVACIONS PER CONTINGUT, ENTRADES DE L'USUARI I COMPROVACIONS PER TIPUS

Mª Belén Tortosa Pedrón

INTRODUCCIÓ

- Les validacions són tècniques que permeten assegurar que els valors amb els quals es vagi a operar estiguin dins de determinat domini.
- Hi ha diferents formes de comprovar el domini d'una dada. Es pot comprovar el contingut, que una variable sigui d'un tipus en particular, o que la dada tingui determinada característica, com que hagi de ser "comparable", o "iterable".
- També s'ha de tenir en compte què farà el nostre codi quan una validació falli, ja que volem donar-li informació al invocant que li serveixi per processar l'error. L'error produït ha de ser fàcilment recognoscible.
- En alguns casos, com per exemple quan es vol tornar una posició, tornar -1 ens pot assegurar que el invocant el vagi a reconèixer. En altres casos, aixecar una excepció és una solució més elegant.

COMPROVACIONS PER CONTINGUT

- Quan volem validar que les dades d'una porció de codi continguin la informació apropiada és desitjable comprovar que el contingut de les variables a utilitzar estiguin dins dels valors amb què es pot operar.
- Per exemple, la funció factorial està definida per als nombres naturals incloent el 0. És possible utilitzar **assert** (que és una altra forma de generar una excepció) per comprovar les precondicions de factorial.

```
def factorial(n):
    """ Calcula el factorial de n.
    Pre: n ha de ser un sencer, major igual a 0
    Post: es retorna el valor del factorial demanat
    """
    assert n >= 0, "n ha de ser major igual a 0"
    fact=1
    for i in xrange(2,n+1):
        fact*=i
    return fact

print(factorial (-8))
```

ASSEVERACIONS

- Tant les precondicions com les postcondicions són asseveracions (en anglès ***assert***). És a dir, afirmacions realitzades en un moment particular de l'execució sobre l'estat computacional. Si arribessin a ser falses significaria que hi ha algun error en el disseny o utilització de l'algoritme.
- Per comprovar aquestes afirmacions des del codi en alguns casos podem utilitzar la instrucció assert, està instrucció rep una condició a verificar i, opcionalment, un missatge d'error que tornarà en cas que la condició no es compleixi. Exemple:

```
>>> n=0
>>> assert n!=0, "El divisor no puede ser 0"
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
AssertionError: El divisor no puede ser 0
```

És important tenir en compte que **assert** està pensat per a ser usat en l'etapa de desenvolupament. Un programa acabat mai hauria de deixar de funcionar per aquest tipus d'errors.

ENTRADA DE L'USUARI

- Per exemple, si es desitja que un usuari, introduïu un número, no s'ha d'assumir que vagi a ingressar-correctament. Ho hem de guardar en una cadena i després convertir a un nombre.

```
def llegeix_sencer():
    """ Sol·licita un valor sencer i el retorna.
        Si el valor teclejat no és sencer,
        llança una excepció."""
    valor = input("Entra un número sencer: ")
    return int(valor)

print(llegeix_sencer())
```

```
C:\WINDOWS\system32\cmd.exe
Entra un número sencer: jj
Traceback (most recent call last):
  File "I:\Curs2016-2017\M3\AWS1\UF2\codi\47_validacions.py", line 7, in <module>
    print(lee_entero())
  File "I:\Curs2016-2017\M3\AWS1\UF2\codi\47_validacions.py", line 5, in lee_entero
    return int(valor)
ValueError: invalid literal for int() with base 10: 'jj'
```

- Aquesta funció retorna un valor sencer, o llança una excepció si la conversió no va ser possible. En el cas en el que l'usuari no hagi ingressat la informació correctament, cal tornar a sol·licitar-la.

EXEMPLE 1: ENTRADA DE DADES

```
def llegeix_sencer():
    """ Sol·licita un valor sencer i el retorna.
        Mentres el valor teclejat no sigui sencer,
        torna a sol·licitar-ho. """
    while True:
        valor = input("Entra un número entero: ")
        try:
            valor = int(valor)
            return valor
        except ValueError:
            print ("ATENCIÓ: Ha de teclejar un número sencer.")

print(llegeix_sencer())
```

```
C:\WINDOWS\system32\cmd.exe
Entra un número entero: i
ATENCIÓ: Ha de teclejar un número sencer.
Entra un número entero: t
ATENCIÓ: Ha de teclejar un número sencer.
Entra un número entero: 5
5
```

EXEMPLE 2: ENTRADA DE DADES

- Podria ser desitjable, a més, posar un límit a la quantitat màxima d'intents que l'usuari té per ingressar la informació correctament i, superada aquesta quantitat màxima d'intents, aixecar una excepció perquè sigui manejada pel codi invocant.

```
def llegeix_sencer():
    """ Sol·licita un valor sencer i el retorna.
        Si el valor teclejat no es sencer, permet 5 intents per
        entrar correctament, de lo contrari, llança un aexcepció
         """

    intents = 0
    while intents < 5:
        valor = input("Entra un número sencer: ")
        try:
            valor = int(valor)
            return valor
        except ValueError:
            intents += 1
    raise ValueError ('Valor incorrecte, has utilitzats 5 intents')

print(llegeix_sencer())
```

```
C:\WINDOWS\system32\cmd.exe
Entra un número sencer: h
Entra un número sencer: j
Entra un número sencer: k
Entra un número sencer: l
Entra un número sencer: m
Traceback (most recent call last):
  File "I:\Curs2016-2017\M3\AWS1\UF2\codi\49_validacions.py", line 17, in <module>
    print(llegeix_sencer())
  File "I:\Curs2016-2017\M3\AWS1\UF2\codi\49_validacions.py", line 15, in llegeix_sencer
    raise ValueError ('Valor incorrecte, has utilitzats 5 intents')
ValueError: Valor incorrecte, has utilitzats 5 intents
```

COMPROVACIONS PER TIPUS

- En aquesta classe de comprovacions ens interessa el tipus de la dada que tractarem de validar. Per exemple, per comprovar que una variable contingui un tipus sencer podem fer:

```
i=input("Entra un numero: ")
if type(i) != int:
    raise TypeError ("i ha de ser de tipus int")
```

```
C:\WINDOWS\system32\cmd.exe
Entra un numero: d
Traceback (most recent call last):
  File "I:\Curs2016-2017\M3\AWS1\UF2\codi\50_validacions.py", line 4, in <module>
    raise TypeError ("i debe ser del tipo int")
TypeError: i debe ser del tipo int
```

COMPROVACIONS PER TIPUS

- No obstant això, ja hem vist que tant les llistes com les tuples i les cadenes són seqüències, i moltes de les funcions utilitzades pot utilitzar qualsevol d'aquestes seqüències. De la mateixa manera, una funció pot utilitzar un valor numèric, i que operi correctament ja sigui sencer, flotant, o complex.
- És possible comprovar el tipus de la nostra variable contra una seqüència de tipus possibles.

```
i=int(input("Entra un valor: "))

if type(i) not in (int, float):
    raise TypeError ("i ha de ser numéric")
```

```
C:\WINDOWS\system32\cmd.exe
Entra un valor: t
Traceback (most recent call last):
  File "I:\Curs2016-2017\M3\AWS1\UF2\codi\51_validacions.py", line 2, in <module>
    i=int(input("Entra un valor: "))
ValueError: invalid literal for int() with base 10: 't'
```

COMPROVACIONS PER TIPUS

- Per a la majoria dels tipus bàsics de Python ha una funció que es diu de la mateixa manera que el tipus que retorna un element d'aquest tipus, per exemple, int () retorna 0, dict () retorna {} i així.
- A més, aquestes funcions solen poder rebre un element d'un altre tipus per intentar convertir-lo, per exemple, int (3.0) retorna 3, list ("Hola") retorna ['H', 'o', 'l', 'a'].

```
def divisio(x,y):
    """ Calcula la divisió sencera després de convertir
    els paràmetres a sencers. """
    try:
        dividendo = int(x)
        divisor = int(y)
        return dividendo/divisor
    except ValueError:
        raise ValueError("x i y s'han de poder convertir a sencers")
    except ZeroDivisionError:
        raise ZeroDivisionError("y no pot ser zero")


print(divisio(3,5))
print(divisio(3,0))
```