prog_datasci_2_python

August 7, 2019

1 Programació per a *Data Science*

1.1 Unitat 2: Breu introducció a la programació en Python

1.1.1 Instruccions d'ús

A continuació es presentarà la sintaxi bàsica del llenguatge de programació Python juntament amb exemples interactius.

1.2 Variables i tipus de variables

Podem entendre una variable com un contenidor en el qual podem posar les nostres dades a fi de guardar-les i tractar-les més endavant. A Python, les variables no tenen tipus, és a dir, no hem d'indicar si la variable serà numèrica, un caràcter, una cadena de caràcters o una llista, per exemple. A més, les variables poden ser declarades i inicialitzades en qualsevol moment, a diferència d'altres llenguatges de programació.

Per declarar una variable, fem servir l'expressió *nom_de_variable = valor*. Es recomana repassar el document PEP-8 per indicar noms de variables correctes, però, *grosso modo*, evitarem utilitzar majúscula a la inicial, separarem les diferents paraules amb el caràcter ń_ż i no utilitzarem accents ni caràcters específics de la nostra codificació com el símbol del ń€ż o la ńñż, per exemple.

Vegem uns quants exemples de declaracions de variables i com fer-les servir:

[1]:
```
# Declarem una variable de nom 'variable_numerica' que conté el valor enter 12.
variable_numerica = 12

# Declarem una variable de nom monstre que conté el valor 'Godzilla'.
monstre = 'Godzilla'

# Declarem una variable de nom 'planetes' que és una llista de cadenes de␣
 ↪caràcters.
planetes = ['Mercuri', 'Venus', 'Terra', 'Mart']
```

[2]:
```
la_meva_edat = 25
la_meva_edat_en_5 = la_meva_edat + 5
# 'Imprimim' el valor calculat que serà, efectivament, 30
print(la_meva_edat_en_5)
```

```
30
```

Els tipus nadius de dades que una variable de Python pot contenir són: nombres enters (int), nombres decimals (float), nombres complexos (complex), cadena de caràcters (string), llistes (list), tuples (tuple) i diccionaris (dict). Vegem un per un cada un d'aquests tipus:

[3]:
```
# Un nombre enter
int_var = 1
another_int_var = -5
# Podem sumar-los, restar-los, multiplicar-los o dividir-los.
print(int_var + another_int_var)
print(int_var - another_int_var)
print(int_var * another_int_var)
print(int_var / another_int_var)

# També podem realitzar la divisió entera.
# Com que només tractem amb nombres enters, no hi haurà part decimal.
print(int_var // another_int_var)
```

```
-4
6
-5
-0.2
-1
```

El comportament de l'operador `/` és una de les diferències entre Python 2 i Python 3. Mentre que en Python 3, l'operador `/` realitza la divisió real entre dos nombres enters (fixeu-vos que `1 / -5` dóna com a resultat `0.2`), en Python 2 realitzava la divisió entera (per la qual cosa el resultat d'executar`1 / -5` en Python 2 seria `-1`). Noteu que usem l'operador `//` per expressar la divisió entera en Python 3.

[4]:
```
# Un nombre decimal o 'float'
float_var = 2.5
another_float_var = .7
# Convertim un nombre enter en un de decimal mitjançant la funció 'float() '
encore_float = float(7)
# Podem fer el mateix en sentit contrari amb la funció 'int()'
new_int = int(encore_float)

# Podem fer les mateixes operacions que en el cas dels nombres enters, però en␣
 ↪aquest cas la divisió serà
# decimal si algun dels nombres és decimal.
print(another_float_var + float_var)
print(another_float_var - float_var)
print(another_float_var * float_var)
print(another_float_var / float_var)
print(another_float_var // float_var)
```

```
3.2
-1.8
1.75
```

```
0.27999999999999997
0.0
```

[5]:
```
# Un nombre complex
complex_var = 2+3j
# Podem accedir a la part imaginària o a la part real:
print(complex_var.imag)
print(complex_var.real)
```

```
3.0
2.0
```

[6]:
```
# Cadena de caràcters
my_string = 'Hello, Bio! ñç'
print(my_string)
```

```
Hello, Bio! ñç
```

Fixeu-vos que podem incloure caràcters unicode (com ñoç) a les cadenes. Això també és una novetat de Python 3 (les variables de tipus `str` són ara UTF-8).

[7]:
```
# Podem concatenar dues cadenes utilitzant l'operador '+'.
same_string = 'Hello, ' + 'Bio' + '!' + ' ñc'
print(same_string)

# A Python també podem utilitzar wildcards com en la funció 'sprintf' de C. Per␣
 ↪exemple:
name = "Guido"
num_emails = 5
print("Hello, %s! You've got %d new emails" % (name, num_emails))
```

```
Hello, Bio! ñc
Hello, Guido! You've got 5 new emails
```

A l'exemple anterior, hem substituït a l'*string* la cadena *%s* pel contingut de la variable *name*, que és un *string*, i *%d* per *num_emails*, que és un nombre enter. També podríem utilitzar *%f* per a nombres decimals (podríem indicar la precisió, per exemple, amb *%5.3f*, el nombre tindria una mida total de cinc xifres i tres serien per la part decimal). Hi ha moltes altres possibilitats, però haurem de tenir en compte el tipus de variable que volem substituir. Per exemple, si utilitzem *%d* i el contingut és *string*, Python retornarà un missatge d'error. Per evitar aquesta situació, es recomana l'ús de la funció *str()* per convertir el valor a *string*.

També podem mostrar el contingut de les variables sense especificar el seu tipus, fent servir `format`:

[8]:
```
print("Hello, {}! You've got {} new emails".format(name, num_emails))
```

```
Hello, Guido! You've got 5 new emails
```

Ara presentarem altres tipus de dades nadius més complexos: llistes, tuples i diccionaris:

[9]:
```
# Definim una llista amb el nom dels planetes (string).
planets = ['Mercury', 'Venus', 'Earth', 'Mars',
           'Jupiter', 'Saturn', 'Uranus', 'Neptune']
# També pot contenir números.
prime_numbers = [2, 3, 5, 7]

# Una llista buida
empty_list = []

# O una barreja de qualsevol tipus:
sandbox = ['3', 'a string', ['a list inside another list', 'second item'], 7.5]
print(sandbox)
```

```
['3', 'a string', ['a list inside another list', 'second item'], 7.5]
```

[10]:
```
# Podem afegir elements a una llista.
planets.append('Pluto')
print(planets)
```

```
['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune',
'Pluto']
```

[11]:
```
# O en podem eliminar.
planets.remove('Pluto')
print(planets)
```

```
['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
```

[12]:
```
# Podem eliminar qualsevol element de la llista.
planets.remove('Venus')
print(planets)
```

```
['Mercury', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
```

[13]:
```
# Sempre que n'afegim, serà al final de la llista. Una llista està ordenada.
planets.append('Venus')
print(planets)
```

```
['Mercury', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune', 'Venus']
```

[14]:
```
# Si volem ordenar-la alfabèticament, podem utilitzar la funció 'sorted()'
print(sorted(planets))
```

```
['Earth', 'Jupiter', 'Mars', 'Mercury', 'Neptune', 'Saturn', 'Uranus', 'Venus']
```

[15]:
```
# Podem concatenar dues llistes:
monsters = ['Godzilla', 'King Kong']
more_monsters = ['Cthulu']
print(monsters + more_monsters)
```

```
['Godzilla', 'King Kong', 'Cthulu']
```

[16]:
```
# Podem concatenar una llista amb una altra i guardar-la a la mateixa llista:
monsters.extend(more_monsters)
print(monsters)
```

```
['Godzilla', 'King Kong', 'Cthulu']
```

[17]:
```
# Podem accedir a un element en concret de la llista:
print(monsters[0])
# El primer element d'una llista és el 0, per tant, el segon serà l'1:
print(monsters[1])
# Podem accedir a l'últim element mitjançant nombres negatius:
print(monsters[-1])
# Penúltim:
print(monsters[-2])
```

```
Godzilla
King Kong
Cthulu
King Kong
```

[18]:
```
# També podem obtenir parts d'una llista mitjançant la tècnica de 'slicing'.
planets = ['Mercury', 'Venus', 'Earth', 'Mars',
           'Jupiter', 'Saturn', 'Uranus', 'Neptune']
# Per exemple, els dos primers elements:
print(planets[:2])
```

```
['Mercury', 'Venus']
```

[19]:
```
# O els elements entre les posicions 2 i 4
print(planets[2:5])
```

```
['Earth', 'Mars', 'Jupiter']
```

Fixeu-vos en aquest últim exemple: en la posició 2 trobem el tercer element de la llista (`'Earth'`) ja que la llista comença a indexar en 0. A més, l'últim element indicat (la posició5) no s'inclou.

[20]:
```
# O els elements del segon al penúltim:
print(planets[1:-1])
```

```
['Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus']
```

La tècnica de ***slicing*** és molt important i ens permet gestionar llistes d'una manera molt senzilla i potent. Serà imprescindible dominar-la per afrontar molts dels problemes que haurem de resoldre en el camp de la ciència de dades.

[22]:
```
# Podem modificar un element en concret d'una llista:
monsters = ['Godzilla', 'King Kong', 'Cthulu']
monsters[-1] = 'Kraken'
print(monsters)
```

```
['Godzilla', 'King Kong', 'Kraken']
```

[23]:
```
# Una tupla és un tipus molt semblant a una llista, però és immutable, és a␣
 ↪dir, un cop declarada
# no podem afegir-hi elements ni eliminar-ne:
birth_year = ('Stephen Hawking', 1942)
# Si executem la línia següent, obtindrem un error de tipus 'TypeError'
birth_year[1] = 1984
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-23-0f1a837a3f5f> in <module>
  3 birth_year = ('Stephen Hawking', 1942)
  4 # Si executem la línia següent, obtindrem un error de tipus␣
↪'TypeError'
----> 5 birth_year[1] = 1984


TypeError: 'tuple' object does not support item assignment
```

Els errors en Python solen ser molt informatius. Una recerca a internet ens ajudarà en la gran majoria de problemes que puguem tenir.

[24]:
```
# Un string també és considerat una llista de caràcters.
# Així doncs, podem accedir a una posició determinada (tot i que no␣
 ↪modificar-la):
name = 'Albert Einstein'
print(name[5])

# Podem fer servir slicing també amb les cadenas de caràcters
print(name[7:15])
```

```
# Podem separar pel caràcter que considerem un string. En aquest cas, per␣
 ↪l'espai en blanc, utilitzant
# la funció 'split()'.
n, surname = name.split()
print(surname)

# I podem convertir un determinat string en una llista de caràcters fàcilment:
chars = list(surname)
print(chars)

# Per unir els diferents elements d'una llista mitjançant un caràcter, podem␣
 ↪utilitzar la funció
# 'join()':
print(''.join(chars))
print('.'.join(chars))
```

```
t
Einstein
Einstein
['E', 'i', 'n', 's', 't', 'e', 'i', 'n']
Einstein
E.i.n.s.t.e.i.n
```

[26]:
```
# L'operador ',' és el creador de tuples. Per exemple, el típic problema␣
 ↪d'assignar el valor d'una
# variable a una altra en Python es pot resoldre en una línia d'una manera molt␣
 ↪elegant utilitzant
# tuples (es tracta d'un idiom):
a = 5
b = -5
a,b = b,a
print(a)
print(b)
```

```
-5
5
```

L'anterior exemple és un *idiom* típic de Python. A la tercera línia, creem una tupla (a,b) a la qual assignem els valors un per un de la tupla (b,a). Els parèntesis no són necessaris, i per això queda una notació tan reduïda.

Per acabar, presentarem els diccionaris, una estructura de dades molt útil a la qual assignem un valor a una clau en el diccionari:

[28]:
```
# Codis internacionals d'alguns països. La clau o 'key' és el codi de país, i␣
 ↪el valor, el seu nom:
country_codes = {34: 'Spain', 376: 'Andorra', 41: 'Switzerland', 424: None}
```

```
# Podem buscar
my_code = 34
country = country_codes[my_code]
print(country)
```

```
Spain
```

[30]:
```
# Podem obtenir totes les claus:
print(country_codes.keys())
```

```
dict_keys([34, 376, 41, 424])
```

[31]:
```
# O els valors:
print(country_codes.values())
```

```
dict_values(['Spain', 'Andorra', 'Switzerland', None])
```

És molt important notar que els valors que obtenim de les claus o en imprimir un diccionari no estan ordenats. És un error molt comú suposar que el diccionari es guarda internament en el mateix ordre en què va ser definit i serà una font d'error habitual no tenir-lo en compte.

[32]:
```
# Podem modificar valors al diccionari o afegir noves claus.

# Definim un diccionari buit. 'country_codes = dict()' és una notació␣
↪equivalent:
country_codes = {}

# Afegim un element:
country_codes[34] = 'Spain'

# N'afegim un altre:
country_codes[81] = 'Japan'

print(country_codes)
```

```
{34: 'Spain', 81: 'Japan'}
```

[33]:
```
# Modifiquem el diccionari:
country_codes[81] = 'Andorra'

print(country_codes)
```

```
{34: 'Spain', 81: 'Andorra'}
```

[34]:
```
# Podem assignar el valor buit a un element:
country_codes[81] = None
```

```
print(country_codes)
```

```
{34: 'Spain', 81: None}
```

Els valors buits ens seran útils per declarar una variable que no sapiguem quin valor o quin tipus de valor contindrà i per fer comparacions entre variables. Habitualment, els valors buits són *None* o '', en el cas de les cadenes de caràcters.

[35]:
```
# Podem assignar el valor d'una variable a una altra. És important que
 ↪s'entenguin les
# línies següents:
a = 5
b = 1
print(a, b)
# b conté la 'direcció' del contenidor al qual apunta 'a'.
b = a
print(a, b)
```

```
5 1
5 5
```

[38]:
```
# Vegem ara què passa si modifiquem el valor d'a o b:
a = 6
print(a, b)
b = 7
print(a, b)
```

```
6 5
6 7
```

Fins aquí hem presentat com declarar i utilitzar variables. Recomanem la lectura de la documentació oficial en línia per a fixar els coneixements explicats.