Notas

Unknown Author

January 9, 2014

1 Primeros pasos Python

Al igual que otros lenguajes de programación Python tiene los tipos usuales de datos: booleanos (bool), enteros (int), flotantes (float) y palabras (strings). Es importante decir que cada uno está definido como un objeto. Esto qué quiere decir, lo que nos quiere decir es que cada uno de ellos tiene definidas ciertas propiedades y métodos para trabajar con ellas.

1.1 Palabras

Podemos definir una palabra y asignarla a una variable de la siguiente manera, el uso de comilla simple puede ser sutituido por comillas dobles sin ningún problema,

```
In [1]: nombre = 'Catalina'

In [2]: apellido = "Valderrama"
```

Para conocer los métodos que pueden ser aplicados en un objeto podemos listarlos así

```
In [3]: dir(str)
Out [3]: ['__add__',
          '__class__',
          '__contains__',
          '__delattr__',
          '__dir__',
          '__doc__',
          '__eq__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__getitem__',
          '__getnewargs__',
          '__gt__',
          '__hash__',
          '__init__',
          '__iter__',
          '__le__',
          '__len__',
          '__lt__',
          '__mod__',
          '__mul__',
```

```
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__rmod__',
'__rmul__',
'__setattr__',
'__sizeof__',
'__str__',
'__subclasshook__',
'capitalize',
'casefold',
'center',
'count',
'encode',
'endswith',
'expandtabs',
'find',
'format',
'format_map',
'index',
'isalnum',
'isalpha',
'isdecimal',
'isdigit',
'isidentifier',
'islower',
'isnumeric',
'isprintable',
'isspace',
'istitle',
'isupper',
'join',
'ljust',
'lower',
'lstrip',
'maketrans',
'partition',
'replace',
'rfind',
'rindex',
'rjust',
'rpartition',
'rsplit',
'rstrip',
'split',
'splitlines',
'startswith',
'strip',
'swapcase',
'title',
'translate',
```

```
 'upper',
 'zfill']
```

Algunos por ejemplo son:

```
In [4]: nombre.upper()
Out [4]: 'CATALINA'
In [5]: nombre.lower()
Out [5]: 'catalina'
```

Para saber que hace un determinado método podemos revisar la ayuda de este.

```
In [6]: help(nombre.find)
Help on built-in function find:

find(...)
    S.find(sub[, start[, end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Return -1 on failure.
```

Además una palabra es una cadena de caractéres (el tipo caracter no está definido en Pyhton), es decir puedo tener nuevas subpalabras de la palabra original

```
In [7]: nombre[1]
Out [7]: 'a'
In [8]: nombre[2:5]
Out [8]: 'tal'
In [9]: nombre[-1]
Out [9]: 'a'
In [10]: nombre[-3:]
Out [10]: 'ina'
In [11]: nombre[-4:-1]
Out [11]: 'lin'
```

La notación `2:5` se lee de la siguiente manera: quiero el pedazo (slide) del tercer caracter (ya que los índices comienzan en 0) hasta antes del quinto espacio entre caractéres.

Ejercicio: Investigue que hacen los demás métodos. Por ejemplo:

```
In [12]: help(nombre.join)
Help on built-in function join:

join(...)
    S.join(iterable) -> str

    Return a string which is the concatenation of the strings in the
    iterable.  The separator between elements is S.
```

```
In [13]: nombre_completo = ' '.join((nombre, apellido))
```

```
In [14]: print(nombre_completo)
         Catalina Valderrama
```

Observación: Muchas veces nos encontraremos con instrucciones como esta

```
In [15]: print('Los hijos de %s tendrán como apellido %s'% (nombre, apellido))
         Los hijos de Catalina tendrán como apellido Valderrama
```

Aún cuando se produce un resultado deseado su uso no es muy *Pythonista*. En su lugar debemos de usar el método `format` para palabras.

```
In [16]: print('Los hijos de {nombre} tendrán como apellido {apellido}'.format(nombre=nombre, a
         Los hijos de Catalina tendrán como apellido Valderrama
```

1.2 Listas

A diferencia de los tipos de datos como enteros, palabras, flotantes o booleanos, las listas son una estructura de datos, es decir es una forma de organizar los datos anteriores de tal manera que sea más fácil manipularlas.

Es importante decir que los elementos de una lista pueden ser de cualquier tipo, inclusive listas.

```
In [17]: mi_lista = [1, 3.5, True, ['a', 'b', 'c']]

In [18]: type(mi_lista)
Out [18]: builtins.list

In [19]: dir(mi_lista)
Out [19]: ['__add__',
           '__class__',
           '__contains__',
           '__delattr__',
           '__delitem__',
           '__dir__',
           '__doc__',
           '__eq__',
           '__format__',
           '__ge__',
           '__getattribute__',
           '__getitem__',
           '__gt__',
           '__hash__',
           '__iadd__',
           '__imul__',
           '__init__',
           '__iter__',
           '__le__',
           '__len__',
           '__lt__',
           '__mul__',
           '__ne__',
           '__new__',
           '__reduce__',
           '__reduce_ex__',
           '__repr__',
           '__reversed__',
           '__rmul__',
           '__setattr__',
```

```
'__setitem__',
'__sizeof__',
'__str__',
'__subclasshook__',
'append',
'clear',
'copy',
'count',
'extend',
'index',
'insert',
'pop',
'remove',
'reverse',
'sort']
```

Es importante hacer enfasis en que para una lista el orden de los elementos importa. Además una manera de ver lo que es una lista es como imaginarse que es una pila de papeles, inclusive muchos de los métodos para listas son tarea que normalmente se hace con las pilas de papeles

```
In [20]: [1, 2, 3, 4] == [1, 2, 4, 3]
Out [20]: False

In [21]: lista_numeros = [1, 2, 3, 4, 5, 6, 7]

In [22]: print(lista_numeros)
         [1, 2, 3, 4, 5, 6, 7]

In [23]: lista_numeros.pop(0)
Out [23]: 1

In [24]: print(lista_numeros)
         [2, 3, 4, 5, 6, 7]

In [25]: lista_numeros.append(1)

In [26]: print(lista_numeros)
         [2, 3, 4, 5, 6, 7, 1]

In [27]: lista_numeros.sort()

In [28]: print(lista_numeros)
         [1, 2, 3, 4, 5, 6, 7]

In [29]: lista_numeros.extend([3, 3, 4, 6, 7, 9])

In [30]: lista_numeros.count(3)
Out [30]: 3

In [31]: print(lista_numeros)
         [1, 2, 3, 4, 5, 6, 7, 3, 3, 4, 6, 7, 9]
```

Ejercicio: ¿Cómo funciona el método `sort` si la lista tiene números y letras?Otra propiedad importante que tienen las listas es que son iterables.

```
In [32]: lista_de_precios_siniva = [12, 34, 56, 23, 75, 92, 45, 97]

In [33]: for precio in lista_de_precios_siniva:
             print('El precio del producto es ${precio:.2f}'.format(precio=precio*1.16))
```

```
El precio del producto es $13.92
El precio del producto es $39.44
El precio del producto es $64.96
El precio del producto es $26.68
El precio del producto es $87.00
El precio del producto es $106.72
El precio del producto es $52.20
El precio del producto es $112.52
```

Ejercicio: ¿Qué hace el siguiente código

```
for indice, precio in enumerate(lista_de_precios_siniva):
    print('El precio del producto (numero) es ${precio:.2f}'.format(numero=indice, precio=p
```

?

¿Estás seguro?. Puedes corregir el código para que haga lo correcto.

Una de las funcionalidades más usadas en las lista es la compresión de las mismas (*list comprehension*). Esto es la capacidad de generar nuevas listas de una lista original.

In [34]:
```
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [35]:
```
cuadrado_primeros = [numero**2 for numero in numeros]
```

In [36]:
```
cuadrado_primeros
```

Out [36]:
```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

In [37]:
```
impares = [numero for numero in numeros if numero%2==1]
```

In [38]:
```
impares
```

Out [38]:
```
[1, 3, 5, 7, 9]
```

In [39]:
```
nombres = ['Juan', 'Jaime', 'Pablo', 'Lucero', 'Jimena', 'Marco']
```

In [40]:
```
nombres_conj = [nombre for nombre in nombres if nombre.startswith('J')]
```

In [41]:
```
print(nombres_conj)
```

```
['Juan', 'Jaime', 'Jimena']
```

In []:
```

```