

DE CERO A CIENCIA DE DATOS



SEMANA 2 - SESIÓN 2 - MARTES 15 DE MAYO

- REVIEW BINARIOS**
- ENTORNO PROGRAMACIÓN**
- EJERCICIOS BÁSICOS PYTHON**

INVOCACIÓN DEL INTÉRPRETE:

```
$ python
```

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 05:52:31)  
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin  
Type "help", "copyright", "credits" or "license" for more information.  
>>>
```

EJECUCIÓN DE PROGRAMA GUARDADO:

```
$ Python nombreprograma.py
```

```
→ Semanal git:(master) ✗ python PythonInterprete090518.py  
alsjfajfsasfjasfjasfjsfjafjfasfjasfjaofjaslfjasfjalffjasfjasfj  
→ Semanal git:(master) ✗
```

CALCULADORA AVANZADA

```
>>> 2+2
```

```
>>> 2+4*8
```

```
>>> (100-20)/6
```

```
>>> (100-20)%6
```

```
>>> (100.0-20)/6
```

```
>>> 2**3
```

```
>>> bin(5)
```

```
>>> oct(87)
```

```
>>> hex(10)
```

EXPRESIONES LÓGICAS

PARA REALIZAR OPERACIONES LÓGICAS TENEMOS LOS OPERADORES AND, OR Y NOT. LOS VALORES QUE DEVUELVEN SON:

EL OPERADOR **AND** DEVUELVE SU PRIMER OPERANDO SI ESTE ES FALSO Y EL SEGUNDO EN CASO CONTRARIO.

EL OPERADOR **OR** DEVUELVE SU PRIMERO OPERANDO SI ESTE ES CIERTO Y EL SEGUNDO EN CASO CONTRARIO.

EL OPERADOR **NOT** DEVUELVE FALSE SI SU OPERANDO ES CIERTO Y TRUE SI ES FALSO.

¿QUÉ ES FALSO?

EL VALOR FALSE.

EL VALOR 0.

UNA SECUENCIA VACÍA (LISTA, TUPLA O CADENA).

UN DICCIONARIO VACÍO.

EL VALOR NONE.

EXPRESIONES LÓGICAS

```
>>> 2 and 4
```

```
>>> 0 and 4
```

```
>>> 2 or 4
```

```
>>> 0 or 4
```

```
>>> not 0
```

```
>>> not 4
```

```
>>> '2' and 4
```

```
>>> '' and 4
```

DELIMITADOR EN CADENAS

LA ELECCIÓN DEL DELIMITADOR SÓLO AFECTA A CÓMO SE ESCRIBEN LAS COMILLAS EN LA CADENA: SI EL DELIMITADOR ES UNA COMILLA DOBLE, LAS COMILLAS DOBLES SE ESCRIBIRÁN ESCAPÁNDOLAS CON UNA BARRA INVERTIDA (\); ANÁLOGAMENTE, SI EL DELIMITADOR ES UNA COMILLA SENCILLA, LAS COMILLAS SENCILLAS DEBERÁN IR ESCAPADAS:

```
>>> "Quiero comprar un CD de Sinéad O'Connor"
```

```
"Quiero comprar un CD de Sinéad O'Connor"
```

```
>>> 'Quiero comprar un CD de Sinéad O\'Connor'
```

```
"Quiero comprar un CD de Sinéad O'Connor"
```

SECUENCIAS DE ESCAPE EN LAS CADENAS

`\` No se incluye en la cadena (sirve para escribir literales de cadena que ocupen más de una línea).

`\\` Barra invertida (backslash).

`\'` Comilla simple.

`\"` Comillas dobles.

`\n` Nueva línea.

OPERADOR % PARA CONCATENAR CADENAS

```
>>> a= 10
```

```
>>> "El resultado es %d" % a
```

```
'El resultado es 10'
```

```
>>> "%d-%d" % (3,6)
```

```
'3-6'
```


CARACTERES DE FORMATO PARA EL OPERADOR % DE CADENAS.

d, i - Entero en decimal.

o - Entero en octal.

x, X - Entero en hexadecimal.

s - Transforma el objeto en cadena usando str.

MÁS CADENAS

```
>>> c="una cadena"
```

```
>>> c.upper()
```

```
'UNA CADENA'
```

```
>>> c.lower()
```

```
'una cadena'
```

Las cadenas son simplemente un tipo especial de secuencias. Todas las secuencias pueden ser "troceadas" utilizando la notación de slices

```
>>> c="cadena"
```

```
>>> c[3]
```

```
'e'
```

```
>>> c[3:5]
```

```
'en'
```

LISTAS

Las listas son secuencias de elementos de cualquier tipo separados por comas. Para escribirlas se utiliza una secuencia de expresiones separadas por comas entre corchetes:

```
>>> s=[1,2,1+1,6/2]
>>> s
[1, 2, 2, 3]
```

Las listas son objetos, así que se pueden invocar métodos sobre ellas:

`insert(i,x)` inserta el elemento `x` en la posición `i`
`append(x)` añade el elemento `x` al final de la lista
`index(x)` devuelve el índice del primer elemento de la lista igual a `x`
`remove(x)` elimina el primer elemento de la lista igual a `x`.
`sort()` ordena los elementos de la lista.
`reverse()` invierte el orden de los elementos de la lista.
`count(x)` cuenta el número de veces que `x` aparece en la lista

TUPLAS

Las tuplas son secuencias inmutables de objetos, es decir no se pueden modificar.

```
>>> a=(1,2,3)
```

```
>>> a[1:3]
```

```
(2, 3)
```

```
>>> a[1]=9
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

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

Los métodos que se pueden invocar son:

count(x) cuenta el número de veces que x aparece en la lista

index(x) devuelve el índice del primer elemento de la lista igual a x

DICCIONARIOS

Python permite utilizar diccionarios. El diccionario vacío se crea mediante {}. Para añadir elementos basta con hacer las correspondientes asignaciones. Después se puede recuperar cualquier elemento a partir de la clave:

```
>>> tel={}
>>> tel["luis"]=1234
>>> tel["maria"]=3456
>>> tel["pepe"]=2323
>>> print tel["maria"]
3456
>>> tel
{'luis': 1234, 'pepe': 2323, 'maria': 3456}
```

items() Regresa todo lo que tiene el dict

keys() Regresa una lista con los keys

values() Regresa una lista con los valores

clear() Elimina todo lo que tiene el diccionario