Python (3)

GOBIERNO DE ESPAÑA
VICEPRESIDENCIA PRIMERA DEL GOBIERNO
MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE DIGITALIZACIÓN E INTELIGENCIA ARTIFICIAL
red.es
Centro de Referencia Nacional en Comercio Electrónico y Marketing
CRN Digital
GARANTÍA JUVENIL
UNIÓN EUROPEA
Barrabés
The Valley
"El FSE invierte en tu futuro"
Fondo Social Europeo

Redondeo

La función **round()** permite redondear un valor real a un número determinado de decimales.

CÓDIGO PYTHON 3

```
n = input('Introduce un número: ')
# Valor original
print(n)
print(type(n))

# Redondeado
n_round = round(float(n),2)
print(n_round)
print(type(n_round))
```

SALIDA

```
Introduce un número: 2.156789
2.156789
<class 'str'>
2.16
<class 'float'>
```

Validación de datos

Podemos utilizar una estructura **`while True … break`** para validar los datos introducidos por el usuario.

CÓDIGO PYTHON 3

```
while True:
    n = int(input('Introduce un número entre 1 y 10: '))
    if n >= 1 and n <= 10:
        break
    else:
        print('Valor incorrecto')
# Aquí continúa el programa
print('Valor correcto')
```

SALIDA

```
Introduce un número entre 1 y
10: 0
Valor incorrecto
Introduce un número entre 1 y
10: 11
Valor incorrecto
Introduce un número entre 1 y
10: 5
Valor correcto
```

Funciones

Funciones

Las funciones permiten reutilizar código.

CÓDIGO PYTHON 3

```
# Función sin parámetros ni retorno
def saludo():
    print('Hola')

n1 = input('Introduce el nombre del usuario 1: ')
saludo()
n2 = input('Introduce el nombre del usuario 2: ')
saludo()
```

SALIDA

```
Introduce el nombre del usuario 1:
Alice
Hola
Introduce el nombre del usuario 2: Bob
Hola
```

Funciones

Las funciones permiten reutilizar código.

CÓDIGO PYTHON 3

```
# Función con parámetros sin retorno
def saludo(nombre):
    print('Hola, ', nombre)

n1 = input('Introduce el nombre del usuario 1: ')
saludo(n1)
n2 = input('Introduce el nombre del usuario 2: ')
saludo(n2)
```

SALIDA

```
Introduce el nombre del usuario 1:
Alice
Hola,  Alice
Introduce el nombre del usuario 2: Bob
Hola,  Bob
```

Funciones

Las funciones permiten reutilizar código.

CÓDIGO PYTHON 3

```
# Función con parámetros y retorno
def saludo(nombre):
    saludo = 'Hola, ' + nombre
    return saludo

n1 = input('Introduce el nombre del usuario 1: ')
print(saludo(n1))
n2 = input('Introduce el nombre del usuario 2: ')
print(saludo(n2))
```

SALIDA

```
Introduce el nombre del usuario 1:
Alice
Hola,  Alice
Introduce el nombre del usuario 2: Bob
Hola,  Bob
```

Módulos

Cómo crear un módulo en Python

Los módulos de Python permiten organizar nuestro código en archivos que después podemos importar y reutilizar en nuestros programas.

Los módulos de Python pueden contener funciones, clases y variables, así como código ejecutable.

Para crear un módulo en Python basta con crear un fichero de texto y guardarlo con la extensión .py.

Para poder utilizar dicho módulo en nuestros programas debemos importarlo con la sentencia import.

Ej.:

Fichero modulo.py:

```
def saludo(nombre):
    print('Hola, ', nombre)
```

Fichero main.py:

```
import modulo
modulo.saludo('Alice')
```

Slicing

Slicing

El slicing consiste en extraer elementos de una secuencia, como una lista o una cadena de caracteres.

CÓDIGO PYTHON 3

```
a = [1,2,3,4,5,6,7,8,9,10]

# a[i] Elementos con índice i
print(a[1])

print()

# a[start:stop] Elementos desde start hasta stop-1
print(a[1:9])

print()

# a[start:] Elementos desde start hasta el final
print(a[1:])

print()

# a[:stop] Elementos desde el principio hasta stop-1
print(a[:9])

print()

# a[:] Todos los elementos
print(a[:])
```

SALIDA

```
2

[2, 3, 4, 5, 6, 7, 8, 9]

[2, 3, 4, 5, 6, 7, 8, 9, 10]

[1, 2, 3, 4, 5, 6, 7, 8, 9]

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Slicing

Los índices de un slicing también pueden ser negativos.

CÓDIGO PYTHON 3

```
a = [1,2,3,4,5,6,7,8,9,10]

# Último elemento
print(a[-1])

print()

# Últimos dos elementos
print(a[-2:])

print()

# Toda la secuencia menos los últimos dos elementos
print(a[:-2])
```

SALIDA

```
10

[9, 10]

[1, 2, 3, 4, 5, 6, 7, 8]
```

Slicing

También es posible indicar un salto (step).

CÓDIGO PYTHON 3

```
a = [1,2,3,4,5,6,7,8,9,10]

# a[start:stop:step] Desde start hasta stop-1 en step
print(a[1:9:2])
```

SALIDA

```
[2, 4, 6, 8]
```

Slicing

Los saltos también pueden ser negativos.

CÓDIGO PYTHON 3

```
a = [1,2,3,4,5,6,7,8,9,10]

# Todos los elementos, invertidos
print(a[::-1])

print()

# Los primeros dos elementos, invertidos
print(a[1::-1])

print()

# Los dos últimos elementos, invertidos
print(a[:-3:-1])

print()

# Todos los elementos menos los dos últimos, invertidos
print(a[-3::-1])
```

SALIDA

```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

[2, 1]

[10, 9]

[8, 7, 6, 5, 4, 3, 2, 1]
```

PYTHON

Estructuras de datos

Estructuras de datos

Estos son los principales tipos de datos estructurados en Python:

- **Listas (list)**
- **Tuplas (tuple)**
- **Conjuntos (set)**
- **Diccionario (dictionary)**

Listas

Una **lista** de Python es una colección **ordenada** de elementos, que pueden ser **del mismo o de diferente tipo**.

Las listas son **mutables** y pueden contener **elementos duplicados**.

La sintaxis para crear una lista es la siguiente:

```
nombre_lista = [elemento_1, elemento_2, …, elemento_n]
```

Listas

Los elementos de una lista pueden ser de cualquier tipo.

CÓDIGO PYTHON 3

```
# Crea una lista de enteros
numeros = [1,2,3,4,5]
# Imprime la lista
print(numeros)
# Accede a un elemento de la lista
print(numeros[1])
# Número de elementos en la lista
print(len(numeros))

print()

# Crea una lista de cadenas
frutas = ['manzana','pera','naranja']
# Imprime la lista
print(frutas)
# Accede a un elemento de la lista
print(frutas[0])
# Número de elementos en la lista
print(len(frutas))
```

SALIDA

```
[1, 2, 3, 4, 5]
2
5

['manzana', 'pera', 'naranja']
manzana
3
```

Listas

Una misma lista puede tener elementos de tipos diferentes, incluso se pueden anidar las listas.

CÓDIGO PYTHON 3

```
# Elementos del mismo tipo
lista_1 = [1,2,3,4,5]
print(lista_1)

print()

# Elementos de diferentes tipos
lista_2 = [False,2.7,'naranja',876,lista_1]
print(lista_2)

print()

# Elementos y tipos de la lista
print(lista_2[0])
print(type(lista_2[0]))
print(lista_2[1])
print(type(lista_2[1]))
print(lista_2[2])
print(type(lista_2[2]))
print(lista_2[3])
print(type(lista_2[3]))
print(lista_2[4])
print(type(lista_2[4]))
```

SALIDA

```
[1, 2, 3, 4, 5]

[False, 2.7, 'naranja', 876, [1, 2, 3,
4, 5]]

False
<class 'bool'>
2.7
<class 'float'>
naranja
<class 'str'>
876
<class 'int'>
[1, 2, 3, 4, 5]
<class 'list'>
```

Listas

Las listas son objetos iterables, de modo que podemos recorrer una lista mediante un bucle **`for`** … **`in`**.

CÓDIGO PYTHON 3

```
# Elementos del mismo tipo
lista_1 = [1,2,3,4,5]
print(lista_1)

print()

# Elementos de diferentes tipos
lista_2 = [False,2.7,'naranja',876,lista_1]
print(lista_2)

print()

# Elementos y tipos de la lista
for elemento in lista_2:
  print(elemento)
  print(type(elemento))
```

SALIDA

```
[1, 2, 3, 4, 5]

[False, 2.7, 'naranja', 876, [1, 2, 3,
4, 5]]

False
<class 'bool'>
2.7
<class 'float'>
naranja
<class 'str'>
876
<class 'int'>
[1, 2, 3, 4, 5]
<class 'list'>
```

Listas

Las listas son objetos que tienen sus propios métodos. Los siguientes son métodos de acceso.

CÓDIGO PYTHON 3

```
# Crea una lista
lista = [3,2,3,1,5,1,3,3,7,8]
print(lista)

print()

# Número de ocurrencias de un elemento
print(lista.count(3))

print()

# Primera ocurrencia de un elemento
print(lista.index(1))
```

SALIDA

```
[3, 2, 3, 1, 5, 1, 3, 3, 7, 8]

4

3
```

Listas

Dado que las listas son objetos mutables, también tienen métodos de modificación.

CÓDIGO PYTHON 3

```
# Crea una lista vacía
lista = []
print(lista)
# Añade elementos a la lista
lista.append(4)
lista.append(2)
lista.append(3)
lista.append(1)
print(lista)
# Ordena la lista
lista.sort()
print(lista)
# Invierte el orden de la lista
lista.reverse()
print(lista)
# Inserta un elemento en una posición
lista.insert(0,3)
print(lista)
# Vacía la lista
lista.clear()
print(lista)
```

SALIDA

```
[]
[4, 2, 3, 1]
[1, 2, 3, 4]
[4, 3, 2, 1]
[3, 4, 3, 2, 1]
[]
```

Tuplas

Al igual que las listas, las tuplas de Python son **colecciones ordenadas** de **elementos heterogéneos**.

La principal diferencia entre las listas y las tuplas es que, mientras que **las listas son dinámicas** (mutables), **las tuplas son estáticas**.

Es decir, **no es posible modificar una tupla una vez se ha creado**.

La sintaxis para crear una tupla es la siguiente:

```
nombre_tupla = (elemento_1, elemento_2, …, elemento_n)
```

Tuplas

Al igual que las listas, los elementos de una tupla pueden ser de cualquier tipo.

CÓDIGO PYTHON 3

```
# Crea una tupla de enteros
numeros = (1,2,3,4,5)
# Imprime la tupla
print(numeros)
# Accede a un elemento de la tupla
print(numeros[1])
# Número de elementos en la tupla
print(len(numeros))

print()

# Crea una tupla de cadenas
frutas = ('manzana','pera','naranja')
# Imprime la tupla
print(frutas)
# Accede a un elemento de la tupla
print(frutas[0])
# Número de elementos en la tupla
print(len(frutas))
```

SALIDA

```
(1, 2, 3, 4, 5)
2
5

('manzana', 'pera', 'naranja')
manzana
3
```

Tuplas

Las tuplas también pueden tener elementos de tipos diferentes e incluso se pueden anidar.

CÓDIGO PYTHON 3

```
# Elementos del mismo tipo
tupla_1 = (1,2,3,4,5)
print(tupla_1)

print()

# Elementos de diferentes tipos
tupla_2 = (False,2.7,'naranja',876,tupla_1)
print(tupla_2)

print()

# Elementos y tipos de la tupla
print(tupla_2[0])
print(type(tupla_2[0]))
print(tupla_2[1])
print(type(tupla_2[1]))
print(tupla_2[2])
print(type(tupla_2[2]))
print(tupla_2[3])
print(type(tupla_2[3]))
print(tupla_2[4])
print(type(tupla_2[4]))
```

SALIDA

```
(1, 2, 3, 4, 5)

(False, 2.7, 'naranja', 876, (1, 2, 3,
4, 5))

False
<class 'bool'>
2.7
<class 'float'>
naranja
<class 'str'>
876
<class 'int'>
(1, 2, 3, 4, 5)
<class 'tuple'>
```

Tuplas

Las tuplas también son objetos iterables, de modo que podemos recorrer una tupla mediante un bucle **`for`** … **`in`**.

CÓDIGO PYTHON 3

```
# Elementos del mismo tipo
tupla_1 = (1,2,3,4,5)
print(tupla_1)

print()

# Elementos de diferentes tipos
tupla_2 = (False,2.7,'naranja',876,tupla_1)
print(tupla_2)

print()

# Elementos y tipos de la lista
for elemento in tupla_2:
  print(elemento)
  print(type(elemento))
```

SALIDA

```
(1, 2, 3, 4, 5)

(False, 2.7, 'naranja', 876, (1, 2, 3,
4, 5))

False
<class 'bool'>
2.7
<class 'float'>
naranja
<class 'str'>
876
<class 'int'>
(1, 2, 3, 4, 5)
<class 'tuple'>
```

Tuplas

Las tuplas son objetos que tienen sus propios métodos, pero como son inmutables solo tienen métodos de acceso.

CÓDIGO PYTHON 3

```
# Crea una tupla
tupla = (3,2,3,1,5,1,3,3,7,8)
print(tupla)

print()

# Número de ocurrencias de un elemento
print(tupla.count(3))

print()

# Primera ocurrencia de un elemento
print(tupla.index(1))
```

SALIDA

```
(3, 2, 3, 1, 5, 1, 3, 3, 7, 8)

4

3
```

Conjuntos

Los conjuntos son colecciones de elementos **no ordenados**.

Los conjuntos **no están indexados** y **no pueden contener elementos duplicados**.

Los elementos de un conjunto **no se pueden cambiar,** pero **se pueden eliminar y añadir**.

La sintaxis para crear un conjunto es la siguiente:

```
nombre_conjunto = {elemento_1, elemento_2, …, elemento_n}
```

Conjuntos

Los elementos de un conjunto pueden ser de cualquier tipo.

CÓDIGO PYTHON 3

```
# Crea un conjunto de enteros
numeros = {1,2,3,4,5}
# Imprime el conjunto
print(numeros)
# Número de elementos en el conjunto
print(len(numeros))

print()

# Crea una conjunto de cadenas
frutas = {'manzana','pera','naranja'}
# Imprime el conjunto
print(frutas)
# Número de elementos en el conjunto
print(len(frutas))
```

SALIDA

```
{1, 2, 3, 4, 5}
5

{'manzana', 'pera', 'naranja'}
3
```

Conjuntos

Un mismo conjunto puede tener elementos de tipos diferentes, pero no se pueden anidar.

CÓDIGO PYTHON 3

```
# Elementos de diferentes tipos
conjunto_1 = {False,2.7,'naranja',876}
print(conjunto_1)

print()

# Conjunto anidado
conjunto_2 = {25, 3.0, conjunto_1}
print(conjunto_2)
```

SALIDA

```
{False, 2.7, 876, 'naranja'}

Traceback (most recent call last):
  File "main.py", line 8, in <module>
    conjunto_2 = {25, 3.0, conjunto_1}
TypeError: unhashable type: 'set'
```

Conjuntos

Un mismo conjunto puede tener elementos de tipos diferentes, pero no se pueden anidar.

CÓDIGO PYTHON 3

```
# Crea un conjunto vacío
conjunto_1 = set()
print(conjunto_1)
print()
# Añade elementos al conjunto
conjunto_1.add('manzana')
conjunto_1.add('pera')
conjunto_1.add('naranja')
print(conjunto_1)
print()
# Crea otro conjunto
conjunto_2 = {'melocoton', 'cereza', 'naranja'}
# Obtiene la interesección
print('Intersección:', conjunto_1.intersection(conjunto_2))
print()
# Obtiene la unión
print('Unión:', conjunto_1.union(conjunto_2))
print()
# Obtiene la diferencia
print('Diferencia:', conjunto_1.difference(conjunto_2))
print()
```

SALIDA

```
set()

{'naranja', 'manzana',
'pera'}

Intersección: {'naranja'}

Unión: {'pera', 'cereza',
'manzana', 'naranja',
'melocoton'}

Diferencia: {'manzana',
'pera'}
```

Diccionarios

Los diccionarios son colecciones de elementos donde los elementos se almacenan de la forma **`clave:valor`**.

Los diccionarios **son mutables** y **no pueden contener elementos duplicados**.

La sintaxis para crear un diccionario es la siguiente:

```
nombre_diccionario = {
    “clave_1”: valor_1,
    “clave_2”: valor_2,
    ...
    “clave_3”: valor_3,
    }
```

Diccionarios

Los elementos de un diccionario pueden ser de cualquier tipo.

CÓDIGO PYTHON 3

```
# Creamos un diccionario
libro = {
     "titulo": "El principito",
     "autor": "Antoine de Saint-Exupéry",
     "num_paginas": 96,
     "es_ebook": False
}

# Imprime el diccionario
print(libro)
print()
# Imprime una clave (campo)
print(libro["titulo"])
print(libro["autor"])
print(libro["num_paginas"])
print(libro["es_ebook"])
print()
# Imprime el número de elementos
print(len(libro))
```

SALIDA

```
{'titulo': 'El principito', 'autor':
'Antoine de Saint-Exupéry',
'num_paginas': 96, 'es_ebook': False}

El principito
Antoine de Saint-Exupéry
96
False

4
```

Diccionarios

Las claves o campos de un diccionario no se pueden repetir. Si lo hacemos sobreescribiremos los valores existentes.

CÓDIGO PYTHON 3

```
# Creamos un diccionario
libro = {
     "titulo": "El principito",
     "autor": "Antoine de Saint-Exupéry",
     "num_paginas": 96,
     "num_paginas": 102,
     "es_ebook": False
}

# Imprime el diccionario
print(libro)
print()
# Imprime una clave (campo)
print(libro["titulo"])
print(libro["autor"])
print(libro["num_paginas"])
print(libro["es_ebook"])
print()
# Imprime el número de elementos
print(len(libro))
```

SALIDA

```
{'titulo': 'El principito', 'autor':
'Antoine de Saint-Exupéry',
'num_paginas': 102, 'es_ebook': False}

El principito
Antoine de Saint-Exupéry
102
False

4
```

Ejercicio en grupo

Diccionarios

Escribe un programa en Python que lleve a cabo las siguientes tareas:

1. Pregunte al usuario si desea introducir datos o realizar una consulta.
2. Si el usuario desea introducir datos se le solicitará la siguiente información y se añadirá a una estructura de datos de tal manera que permita su posterior consulta por DNI:
    - DNI
    - Nombre
    - Apellido 1
    - Apellido 2
3. Si el usuario desea realizar una consulta se le solicitará que introduzca el DNI y se buscará dicho DNI en la estructura datos creada. Si se encuentra una entrada con el DNI indicado se mostrará su nombre y apellidos. Si no se encuentra ninguna entrada con ese DNI se informará al usuario de que no existe.
4. Una vez introducidos los datos o realizada la consulta, el programa deberá volver al paso 1.

Cuestiones extra:

- Añade una función que permita eliminar entradas a partir de su DNI.
- Cada vez que iniciamos el programa la estructura de datos está vacía. ¿Cómo podemos hacer que sea persistente?

red.es

CRN Digital

GARANTÍA JUVENIL

UNIÓN EUROPEA

"El FSE invierte en tu futuro"

Fondo Social Europeo

Barrabés

The Valley