

## A) Tipos de datos compuestos en Python

### 1. Lista, array o vector

Es una secuencia de datos ordenada, heterogénea y mutable, que se escriben entre corchetes y están separados por comas.

- heterogénea: puede contener diferentes tipos de datos
- mutable: sus elementos pueden modificarse
- ordenada: podemos acceder a cada uno de sus elementos por medio de un índice, siendo 0 el índice del primer elemento. También podemos utilizar índices negativos, siendo -1 el último elemento de la lista.

```
mi_lista = ['pan', 32, True, 20.5, [0, 'Juan']]
print(mi_lista) # ['pan', 32, True, 20.5, [0, 'Juan']]

print(mi_lista[0]) # pan
print(mi_lista[2]) # True

mi_lista[2] = False
print(mi_lista[2]) # False

print(mi_lista[-1]) # [0, 'Juan']
```

### Función len()

Devuelve la longitud (cantidad de elementos) de una lista

```
mi_lista = ['pan', 32, True, 20.5, [0, 'Juan']]
print(len(mi_lista)) # 5
```

## Métodos

- **append():** agrega un elemento al final de una lista
- **count():** recibe un elemento como argumento, y cuenta la cantidad de veces que aparece en la lista
- **extend():** extiende una lista agregando elementos
- **index():** recibe un elemento como argumento y devuelve el índice de su primera aparición en la lista. Opcionalmente se pueden agregar argumentos adicionales para indicar en qué índices iniciar y terminar la búsqueda. El método devuelve ValueError si no encuentra el elemento en la lista.
- **insert():** inserta un elemento en un índice determinado
- **pop():** muestra el último elemento y lo borra de la lista. Opcionalmente puede recibir un argumento como índice del elemento a eliminar (por defecto es -1).
- **remove():** recibe como argumento un elemento y borra su primera aparición de la lista. El método devuelve ValueError si no encuentra el elemento en la lista.
- **reverse():** invierte el orden de los elementos de una lista
- **sort():** ordena los elementos de una lista

```
mi_lista = ['pan', 32, True, 20.5, [0, 'Juan']]
# append()
mi_lista.append('Toyota')
print(mi_lista) # ['pan', 32, True, 20.5, [0, 'Juan'], 'Toyota']
```

```
# count()
lista2 = [2, 3, 2.6, 2, 6, 2, 5, 6]
print(lista2.count(2)) # 3
```

```
lista2 = [2, 3, 2.6, 2, 6, 2, 5, 6]
# extend
lista2.extend([4])
print(lista2) # [2, 3, 2.6, 2, 6, 2, 5, 6, 4]

lista2.extend(range(2,8,2))
print(lista2) # [2, 3, 2.6, 2, 6, 2, 5, 6, 4, 2, 4, 6]
```



```
lista2 = [2 , 3, 2.6, 2, 6, 2, 5, 6]
# index()
print(lista2.index(2)) # 0
print(lista2.index(2, 1)) # 3
print(lista2.index(22, 1)) # ValueError: 22 is not in list
```

```
lista2 = [2 , 3, 2.6, 2, 6, 2, 5, 6]
# insert()
lista2.insert(3, 1000)
print(lista2) # [2, 3, 2.6, 1000, 2, 6, 2, 5, 6]
```

```
lista2 = [2 , 3, 2.6, 2, 6, 2, 5, 6]
# pop()
print(lista2.pop()) # 6
print(lista2) # [2 , 3, 2.6, 2, 6, 2, 5]
print(lista2.pop(2)) # 2.6
print(lista2) # [2, 3, 2, 6, 2, 5]
```

```
lista2 = [2 , 3, 2.6, 2, 6, 2, 5, 6]
# reverse()
lista2.reverse()
print(lista2) # [6, 5, 2, 6, 2, 2.6, 3, 2]
# sort()
lista2.sort()
print(lista2) # [2, 2, 2, 2.6, 3, 5, 6, 6]
```

## Conversión a tipo lista

Podemos convertir a tipo de dato lista, mediante la función list()

```
lista3 = list(range(11))  
print(lista3) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

## 2. Tuplas

Es una secuencia de datos ordenada, heterogénea e inmutable, que se escriben entre paréntesis y están separados por comas.

- heterogénea: puede contener diferentes tipos de datos
- inmutable: sus elementos no pueden modificarse después de su creación
- ordenada: podemos acceder a cada uno de sus elementos por medio de un índice, siendo 0 el índice del primer elemento. También podemos utilizar índices negativos, siendo -1 el último elemento de la lista.

### Función len()

Devuelve la longitud (cantidad de elementos) de una tupla

### Métodos

- **count()**: recibe un elemento como argumento, y cuenta la cantidad de veces que aparece en la tupla
- **index()**: recibe un elemento como argumento y devuelve el índice de su primera aparición en la tupla. Opcionalmente se pueden agregar argumentos adicionales para indicar en qué índices iniciar y terminar la búsqueda. El método devuelve ValueError si no encuentra el elemento en la lista.

```
mi_tupla = ('Python', False, 33, 'Santiago', False)  
print(len(mi_tupla)) # 5  
print(mi_tupla.count(False)) # 2  
print(mi_tupla.index('Python')) # 0
```

### Conversión a tipo tupla

Podemos convertir a tipo de dato tupla, mediante la función tuple()



```
tupla2 = tuple(range(4,12))  
print(tupla2) # (4, 5, 6, 7, 8, 9, 10, 11)
```

### 3. Diccionarios

Son mapeos de datos desordenados y mutables, que se escribe entre llaves, están organizados por pares “key: value” y separados por comas.

Se puede acceder a los valores del diccionario a través de su clave (key).

```
diccionario = {  
    'Nombre' : 'Ana',  
    'Apellido' : 'Alvarez',  
    'Edad' : 27,  
    'Trabaja' : True,  
    'Notas' : [8, 9, 7]  
}  
print(diccionario['Nombre']) # Ana  
print(type(diccionario['Notas'])) # <class 'list'>
```

Existe una manera alternativa de crear diccionarios con el método dict():

```
dict2 = dict(lenguaje = 'Python', version = 3.10, framework = 'Flask')  
print(dict2) # {'lenguaje': 'Python', 'version': 3.1, 'framework': 'Flask'}
```

### Operaciones

- Acceder al valor de una clave: mediante su clave.
- Asignar valor a una clave
- Iteración in: devuelve True si una clave está en el diccionario

```
dict2 = dict(lenguaje = 'Python', version = 3.10, framework = 'Flask')
# Acceder al valor de clave
print(dict2['lenguaje']) # Python

# Asignar valor a una clave
dict2['framework'] = 'Django'
print(dict2) # {'lenguaje': 'Python', 'version': 3.1, 'framework': 'Django'}

# Iteración in
print('version' in dict2) # True
print('apellido' in dict2) # False
```

## Métodos

- **clear():** remueve todos los elementos de un diccionario
- **copy():** devuelve una copia del diccionario
- **fromkeys():** crea un nuevo diccionario con claves a partir de un tipo de dato secuencia. El valor por defecto es de tipo None.
- **get():** devuelve el valor de una búsqueda mediante clave. None si no lo encuentra.
- **items():** Devuelve una lista de tuplas.
- **keys():** Devuelve una lista con las claves del diccionario.
- **pop():** Remueve una clave del diccionario. Lanza error KeyError si no la encuentra.
- **popitem():** Remueve un par clave:valor del diccionario como 2-tupla
- **setdefault():** asigna valores por defecto a las claves de un diccionario.
- **update():** actualiza un diccionario agregando los pares clave:valor en un segundo diccionario.
- **values():** Devuelve una lista de los valores del diccionario.

```
dict1 = dict(lenguaje = 'Python', version = 3.10, framework = 'Flask')
# copy()
dict2 = dict1.copy()
print(dict2) # {'lenguaje': 'Python', 'version': 3.1, 'framework': 'Flask'}

# clear()
dict2.clear()
print(dict2) # {}
```

```
# fromkeys()
tupla = ('nombre', 'apellido', 'edad')
dict3 = dict.fromkeys(tupla)
print(dict3) # {'nombre': None, 'apellido': None, 'edad': None}
```



```
dict4 = dict(alumno = 235645, nota1 = 8.50, nota2 = 9.75, nota3 = 7.25)
# get()
print(dict4.get('nota2')) # 9.75
print(dict4.get('apellido')) # None

# items()
print(dict4.items())
# dict_items([('alumno', 235645), ('nota1', 8.5), ('nota2', 9.75), ('nota3', 7.25)])
```

```
dict4 = dict(alumno = 235645, nota1 = 8.50, nota2 = 9.75, nota3 = 7.25)
# keys()
print(dict4.keys()) # dict_keys(['alumno', 'nota1', 'nota2', 'nota3'])

# pop()
print(dict4.pop('alumno')) #235645
print(dict4) # {'nota1': 8.5, 'nota2': 9.75, 'nota3': 7.25}

dict4 = dict(alumno = 235645, nota1 = 8.50, nota2 = 9.75, nota3 = 7.25)
# popitem()
print(dict4.popitem()) # ('nota3', 7.25)
print(dict4) # {'alumno': 235645, 'nota1': 8.5, 'nota2': 9.75}
```

```
dict4 = dict(alumno = 235645, nota1 = 8.50, nota2 = 9.75, nota3 = 7.25)
# setdefault()
alumno = dict4.setdefault('alumno') # la clave existe
print(alumno) # 235645
apellido = dict4.setdefault('apellido') # la clave no existe
print(apellido) # None
nombre = dict4.setdefault('nombre', 'Eduardo') # la clave no existe, pero damos dato
print(nombre) # Eduardo
print(dict4)
# {'alumno': 235645, 'nota1': 8.5, 'nota2': 9.75, 'nota3': 7.25,
# 'apellido': None, 'nombre': 'Eduardo'}
```

```
dict4 = dict(alumno = 235645, nota1 = 8.50, nota2 = 9.75, nota3 = 7.25)
# update
otro_dict = dict(nombre = 'Álvaro')
dict4.update(otro_dict)
print(dict4)
# {'alumno': 235645, 'nota1': 8.5, 'nota2': 9.75,
# 'nota3': 7.25, 'nombre': 'Álvaro'}

# values()
print(dict4.values()) # dict_values([235645, 8.5, 9.75, 7.25, 'Álvaro'])
```

## Funciones

- **len()**: devuelve la cantidad de elementos de un diccionario.

```
dict5 = dict(nota1 = 8.50, nota2 = 9.75, nota3 = 7.25)

# len()
print(len(dict5)) # 3
```

## 4. Conjuntos

Un conjunto es una colección no ordenada y sin elementos repetidos. Los usos básicos de este tipo de datos son la verificación de pertenencia y eliminación de entradas duplicadas.

### Set

Es de tipo mutable, desordenado y no contiene duplicados.

### Frozenset

Es de tipo inmutable, desordenado y no contiene duplicados.

### Métodos

- **add()**: agrega un elemento a un conjunto mutable.



- **clear():** remueve todos los elementos de un conjunto mutable.
- **copy():** devuelve una copia de un conjunto mutable o inmutable.
- **difference():** devuelve la diferencia entre dos conjuntos mutables o inmutables.
- **difference\_update():** actualiza un tipo conjunto mutable con la diferencia de los conjuntos.
- **discard():** remueve un elemento de un conjunto mutable.
- **intersection():** devuelve la intersección entre los conjuntos mutables o inmutables.
- **intersection\_update():** actualiza un conjunto mutable con la intersección de ese mismo y otro conjunto mutable.
- **isdisjoint():** devuelve True si no hay elementos comunes entre conjuntos mutables o inmutables.
- **issubset():** devuelve True si el conjunto mutable es un subconjunto del conjunto mutable o inmutable del argumento.
- **issuperset():** devuelve True si el conjunto mutable o inmutable es un superset (contiene) del conjunto mutable argumento.
- **pop():** elimina aleatoriamente un elemento del conjunto mutable.
- **remove():** elimina arbitrariamente un elemento de un conjunto mutable.
- **symmetric\_difference():** devuelve todos los elementos que están en un conjunto mutable e inmutable u otro, pero no en ambos.
- **symmetric\_difference\_update():** actualiza a un conjunto mutable con el contenido de la diferencia simétrica.
- **union():** devuelve un conjunto mutable e inmutable con todos los elementos que están en alguno de los conjuntos mutable e inmutables
- **update():** agrega elementos desde un conjunto mutable pasado como argumento.

```
mi_set = set([4, 3, 11, 7, 5, 2, 1, 4])
print(mi_set) # {1, 2, 3, 4, 5, 7, 11}

# add()
mi_set.add(22)
print(mi_set) # {1, 2, 3, 4, 5, 7, 11, 22}

# copy()
mi_set2 = mi_set.copy()
print(mi_set == mi_set2) # True

# clear()
mi_set2.clear()
print(mi_set2) # set()
```

<codoa  
codo/>

```
mi_set = set([3, 11, 7, 5, 2, 1, 4, 23])
mi_set2 = set([3, 11, 7, 5, 2, 1, 4, 55, 70])

# difference
print(mi_set.difference(mi_set2)) # {23}
print(mi_set2.difference(mi_set)) # {70, 55}

# difference_update
mi_set2.difference_update(mi_set)
print(mi_set2) # {70, 55}

# discard
mi_set.discard(23)
print(mi_set) # {1, 2, 3, 4, 5, 7, 11}
```

```
mi_set = set([3, 11, 7, 5, 2, 1, 4, 23])
mi_set2 = set([3, 11, 7, 5, 2, 1, 4, 55, 70])

# intersection()
print(mi_set.intersection(mi_set2)) # {1, 2, 3, 4, 5, 7, 11}

# intersection_update()
mi_set.intersection_update(mi_set2)
print(mi_set) # {1, 2, 3, 4, 5, 7, 11}
```



<codoa  
codo/>

```
mi_set = set([3, 11, 7, 5, 2, 1, 4])
mi_set2 = set([3, 11, 7, 5, 2, 1, 4, 55, 70])

# isdisjoint()
print(mi_set.isdisjoint(mi_set2)) # False

# issubset()
print(mi_set.issubset(mi_set2)) # True

# issuperset()
print(mi_set2.issuperset(mi_set)) # True
```

```
mi_set = set([3, 11, 7, 5, 2, 1, 4])

# pop()
print(mi_set.pop()) # 1
print(mi_set) # {2, 3, 4, 5, 7, 11}

# remove()
mi_set.remove(7)
print(mi_set) # {2, 3, 4, 5, 11}

mi_set2 = { 55, 112}
# update()
mi_set.update(mi_set2)
print(mi_set) # {2, 3, 4, 5, 11, 112, 55}
```

```
mi_set = set([3, 11, 7, 5, 2, 1, 4, 55, 77])
mi_set2 = set([3, 11, 7, 5, 2, 1, 4, 100, 123])

# symmetric_difference()
print(mi_set.symmetric_difference(mi_set2)) # {100, 77, 55, 123}

# union()
print(mi_set.union(mi_set2)) # {1, 2, 3, 4, 5, 100, 7, 11, 77, 55, 123}

# symmetric_difference_update()
mi_set.symmetric_difference_update(mi_set2)
print(mi_set) # {100, 77, 55, 123}
```

## B) Slicing: rebanadas

Un slice es un subconjunto en una lista de elementos. La función slice() devuelve un objeto slice (una rebanada de una lista o tupla).

### Sintaxis

x = slice(inicio, final, step)

```
a = [2, 5, 8, 11, 15, 18, 20, 22, 50]
x = slice(2,9,3)
print(a[x]) # [8, 18, 50]
```

### Notación

a[inicio:final] # desde el elemento 'inicio' hasta 'final'-1  
a[inicio:] # desde el elemento 'inicio' hasta el final del array  
a[:final] # desde el primer elemento hasta elemento 'final'-1  
a[:] # todos los elementos del array  
a[-1] # selecciona el último elemento del array  
a[-2:] # selecciona los dos últimos elementos del array



`a[:-2]` # selecciona todos los elementos excepto los dos últimos

```
a = [2, 5, 8, 11, 15, 18, 20, 22]
print(a[1:4]) # [5, 8, 11]
print(a[5:]) # [18, 20, 22]
print(a[:4]) # [2, 5, 8, 11]
print(a[:]) # [2, 5, 8, 11, 15, 18, 20, 22]
print(a[-1]) # 22
print(a[-2:]) # [20, 22]
print(a[:-2]) # [2, 5, 8, 11, 15, 18]
```

## C) Funciones Built-in in Python

El intérprete de Python tiene disponible una serie de funciones y tipos. Algunas de las más utilizadas:

- **dict(x):** crea un nuevo diccionario.
- **format(valor[, especificaciones]):** convierte a el valor a una representación con formato, según especificaciones.
- **frozenset([iterable]):** devuelve un objeto frozenset, opcionalmente con elementos del iterable.
- **help([objeto]):** invoca el sistema de ayuda integrado de Python.
- **hex(x):** convierte un número entero a una cadena hexadecimal con prefijo "0x".
- **id(object):** retorna la "identidad" de un objeto.
- **input([mensaje]):** Lee lo que se ingresa por teclado. Opcionalmente se puede imprimir una línea de mensaje.
- **len(s):** devuelve el tamaño (cantidad de elementos) de un objeto (cadena, tupla, lista, rango, etc.).
- **list([iterable]):** tipo de secuencia mutable.
- **map(función, iterable, ...):** devuelve un iterador que aplica la función a cada uno de los elementos.
- **max/min(iterable):** devuelve el máximo/mínimo elemento de un iterable o de un grupo de dos o más argumentos.
- **oct(x):** convierte un número entero a una cadena octal con prefijo "0o".
- **print(obj):** imprime por pantalla uno o varios objetos.
- **range(start, stop[, step]):** tipo de secuencia inmutable.

- **reversed(seq):** devuelve una secuencia en orden inverso.
- **set([iterable]):** retorna un objeto de tipo set, opcionalmente con elementos tomados de iterable.
- **slice(start, stop[, step]):** retorna un objeto slice que representa el conjunto de índices especificados por range.
- **sorted(iterable):** retorna una nueva lista ordenada a partir de los elementos en iterable.
- **str(objeto):** retorna una versión str de objeto.
- **sum(iterable, start=0):** Suma start y los elementos de un iterable de izquierda a derecha y retorna el total.
- **tuple([iterable]):** tipo de secuencia inmutable.
- **type(objeto):** devuelve el tipo del objeto.

## Para números enteros y flotantes

- **abs(x):** retorna el valor absoluto del número x. x debe ser un número entero o de punto flotante.
- **divmod(a, b):** toma dos números como argumentos y retorna un par de números que serán el cociente y el resto de la división entera.
- **float(x):** convierte un número o una cadena x en número de punto flotante.
- **int(x):** convierte un número o cadena x a un número entero.
- **pow(base, exponente):** retorna el valor de la base elevado a exponente. Es equivalente a  $base^{exponente}$ .
- **round(nro[, ndígitos]):** devuelve nro redondeado a ndígitos.

Fuente: <https://docs.python.org/es/3.8/library/functions.html>

## D) Métodos de las cadenas (str)

### Métodos de análisis

- **count():** devuelve el número de veces que se repite un conjunto de caracteres especificado
- **find(), index():** devuelven la ubicación en la que se encuentra el argumento indicado.
- **rfind(), rindex():** busca cadenas de caracteres empezando por la derecha.



- **startswith(), endswith():** devuelve True o False si la cadena comienza o termina con el conjunto de caracteres que se pasa como argumento.
- **isalnum():** determina si todos los caracteres son alfanuméricos.
- **isalpha():** determina si todos los caracteres son alfabéticos
- **isdigit(), isnumeric(), isdecimal():** determinan si todos los caracteres de la cadena son dígitos, números o números decimales.
- **islower(), isupper():** determina si todos los caracteres están en minúsculas o mayúsculas.
- **isspace():** determina si una cadena tiene sólo espacios.
- **isprintable():** determina si todos los caracteres de la cadena son imprimibles.

## Métodos de transformación

- **capitalize():** convierte la primera letra de la cadena a mayúscula.
- **center(), ljust(), rjust():** alinean una cadena en el centro, a la izquierda o a la derecha. Toman como argumento la cantidad de caracteres respecto de la cual se producirá la alineación. Un segundo argumento indica con qué carácter queremos llenar los espacios (por defecto, en blanco)
- **lower(), upper():** devuelven una copia de la cadena con todas las letras en minúsculas o mayúsculas.
- **swapcase():** cambia mayúsculas por minúsculas y viceversa.
- **strip(), lstrip(),rstrip():** eliminan los espacios en blanco que preceden o suceden a la cadena.
- **replace():** reemplaza una cadena por otra.

## Métodos de separación y unión

- **split():** separa una cadena y los convierte en lista. Los separadores por defecto son los espacios y los saltos de línea. Puede indicarse otro separador como argumento
- **partition():** devuelve una tupla de 3 elementos: el bloque de caracteres anterior a la primera aparición del separador, el separador, y el bloque posterior.
- **rpartition():** similar al anterior, pero empezando desde la derecha.
- **join():** une elementos de una lista, utilizando una cadena como separador.

## E) Funciones matemáticas: math

El intérprete de Python ofrece funciones matemáticas por medio del módulo **math**.

Podemos importar este módulo con la línea:

```
import math
```

Algunas de las funciones más utilizadas son:

- **math.ceil(x)**: redondea hacia arriba.
- **math.factorial(x)**: devuelve el factorial de x.
- **math.floor(x)**: redondea hacia abajo.
- **math.sum(iterable)**: devuelve una suma precisa con coma flotante de los valores de un iterable.
- **math.gcd(\*enteros)**: devuelve el mcd de la serie de argumentos.
- **math.lcm(\*enteros)**: devuelve el mcm de la serie de argumentos.
- **math.isqrt(x)**: devuelve la raíz cuadrada de un número entero no negativo. (entero)
- **math.trunc(x)**: devuelve x con la parte fraccionaria eliminada.

## Funciones logarítmicas y exponenciales

- **math.exp(x)**: retorna e elevado a la x potencia, donde  $e = 2,718281...$
- **math.expm1(x)**: retorna e elevado a la x potencia, menos 1.
- **math.log(x[, base])**: con un argumento, retorna el logaritmo natural de x (en base e). Con dos argumentos, retorna el logaritmo de x en la base dada.
- **math.log1p(x)**: retorna el logaritmo natural de  $1+x$ .
- **math.log2(x)**: retorna el logaritmo en base 2 de x.
- **math.log10(x)**: retorna el logaritmo en base 10 de x.
- **math.pow(x, y)**: retorna x elevado a la potencia y.
- **math.sqrt(x)**: retorna la raíz cuadrada de x. (flotante)

También están disponibles las funciones hiperbólicas **acosh**, **asinh**, **atanh**, **cosh**, **sinh** y **tanh**.

## Funciones trigonométricas

- **math.acos(x)**: retorna el arcocoseno de x en radianes.  $[0, \pi]$
- **math.asin(x)**: retorna el arcoseno de x, en radianes.  $[-\pi/2, \pi/2]$



- **math.atan(x)**: retorna la arcotangente de x en radianes.  $[-\pi/2, \pi/2]$
- **math.atan2(x, y)**: retorna atan(y / x), en radianes  $[-\pi, \pi]$
- **math.cos(x)**: retorna el coseno de x en radianes.
- **math.sin(x)**: retorna el seno de x en radianes.
- **math.tan(x)**: retorna la tangente de x en radianes.

## Conversión angular

- **math.degrees(x)**: convierte el ángulo x de radianes a grados.
- **math.radians(x)**: convierte el ángulo x de grados a radianes.

## Constantes

- **math.pi**
- **math.e**
- **math.tau**
- **math.inf**: valor infinito positivo en punto flotante.
- **math.nan**: un valor de coma flotante de tipo “no es un número”.

Fuente: <https://docs.python.org/es/3/library/math.html#module-math>

## F) El módulo random

Este módulo incluye un conjunto de funciones que nos permiten obtener números aleatorios.

Podemos importar este módulo con la línea:

```
import random [as alias]
```

- **random.randint()**: devuelve un número entero incluido entre los valores indicados. Los valores de los límites inferior y superior también pueden aparecer entre los valores devueltos.

# <codoa codoo/>

```
import random as r

candidatos = ['Ana', 'Juan', 'Daniel', 'Laura', 'José', 'Clara']
for sorteo in range(3):
    ganador = candidatos[r.randint(0,5)]
    print('Ganador', sorteo + 1, ':', ganador)
```

```
import random as r

# imprime 10 nros enteros entre -5 y 20
for numero in range(5):
    print(r.randint(-10,20))
```

- **random.randrange():** devuelve enteros que van desde un valor inicial a otro final separados por un paso.

```
import random as r

# imprime 5 múltiplos de 4, separados por ' '
for i in range(5):
    print(r.randrange(4,50,4), end = ' ')
```

- **random.random():** devuelve un número float entre 0 y 1.

```
import random as r

for i in range(5):
    print(r.random())

...
0.8992764679053034
0.0740968548328993
0.8180031166838229
0.40850380396017527
0.6422579590270423
...
```

Agencia de  
Aprendizaje  
a lo largo  
de la vida



- **random.uniform()**: devuelve un número float incluido entre los valores indicados.
- **random.seed()**: se utiliza cuando queremos obtener varias veces la misma secuencia de números. El siguiente ejemplo, elige 3 animales, y se repite la misma selección cada vez que ejecutamos ese bloque.

```
import random as r

candidatos = ['perro', 'gato', 'hamster', 'tortuga', 'canario', 'pez', 'boa']
for mascota in range(2):
    print('Sorteo', mascota + 1)
    r.seed(0)
    for sorteo in range(3):
        ganador = candidatos[r.randint(0, 6)]
        print('Elegido', sorteo + 1, ':', ganador)
```

- **random.choice()**: selecciona elementos al azar de una lista.

```
import random as r

candidatos = ['perro', 'gato', 'hamster', 'tortuga', 'canario']
print('Voy a adoptar: ', r.choice(candidatos))
```

- **random.shuffle()**: mezcla o cambia aleatoriamente el orden de los elementos de una lista antes de realizar la selección.

```
import random as r

palos = ['bastos', 'copas', 'oros', 'espadas']
valor = ['As', 2, 3, 4, 5, 6, 7, 8, 9, 'sota', 'caballo', 'rey']
r.shuffle(palos)
r.shuffle(valor)
print('Mezcla 1 ', palos, valor)
r.shuffle(palos)
r.shuffle(valor)
print('Mezcla 2 ', palos, valor)
print('\nSale', valor[r.randint(0,11)], 'de', palos[r.randint(0,3)])
```

- **random.sample():** devuelve de una lista de elementos un determinado número de elementos diferentes elegidos al azar.

```
import random as r

valor = ['As',2,3,4,5,6,7,8,9,'sota','caballo','rey']
print(r.sample(valor,5))
```

## G) f-strings

Es una nueva forma de aplicar formato a cadenas de texto que surgió a partir de Python 3.6.

Las f-strings o “cadenas literales”, son cadenas de texto precedidas por una f antes de las comillas de apertura, que pueden contener nombres variables (encerradas entre llaves), que al momento de imprimirse, van a mostrar su valor actual.

Veamos algunos ejemplos:

```
nombre = 'Ramón'
edad = 54
print(f'{nombre} tiene {edad} años')
# Ramón tiene 54 años
```

```
# MÚLTIPLES RENGLONES I
nombre = 'Ana'
apellido = 'López'
edad = 35
profesion = 'abogada'
mensaje = (
    f'Nombre: {nombre} - '
    f'Apellido: {apellido} - '
    f'Edad: {edad} - '
    f'Profesión: {profesion}'
)
print(mensaje)
# Nombre: Ana - Apellido: López - Edad: 35 - Profesión: abogada
```





<codoa  
codoo/>

```
# CONDICIONALES
nombre = input('Nombre del alumno? ')
promedio = float(input(f'Ingrese el promedio de {nombre}: '))
print(f'¿{nombre} está aprobado? {"SI" if promedio >= 6 else "NO"}')
'''Nombre del alumno? Juan
Ingrese el promedio de Juan: 8
¿Juan está aprobado? SI'''
```

```
# EXPRESIONES
n1 = 23
print(f'{n1} + 1000') # 23 + 1000
print(f'{n1 + 1000}') # 1023

# FUNCIONES
def minuscula(texto):
    return texto.lower()

palabra = 'PERRITO'
print(f'Texto en minúsculas: {minuscula(palabra)}')
# Texto en minúsculas: perrito
```

## H) Fechas en Python

Las fechas en Python no son un tipo de dato, pero podemos importar el módulo `datetime` para trabajarlas como objetos.

```
import datetime

hoy = datetime.datetime.now()
print(hoy) # 2022-04-28 23:48:08.986204
```

Agencia de  
Aprendizaje  
a lo largo  
de la vida

## Mostrar fechas

Para ver el año y nombre del día de la semana:

```
import datetime

hoy = datetime.datetime.now()

print(hoy.year) # 2022
print(hoy.strftime("%A")) # Thursday
```

## Creando fechas

```
import datetime

fecha = datetime.datetime(2022, 5, 14)
print(fecha) # 2022-05-14 00:00:00
```

## El método strftime()

Permite formatear los objetos date en cadenas. Utiliza un parámetro para especificar el formato (consulta la tabla en el documento 'Códigos de formato strptime y strftime.pdf' de esta unidad).

Por ejemplo, para mostrar el nombre del mes:

```
import datetime

fecha = datetime.datetime(2022, 5, 14)
print(fecha.strftime('%B')) # May
```



## I) Objetos mutables e inmutables

En Python todos los tipos de datos se manejan como objetos, entre ellos: cadenas, listas, tuplas, etc.

Estos tipos de datos ya los hemos definido en su momento como mutables o inmutables.

Pero ¿qué significa esto exactamente? Simplemente, si el objeto puede cambiar, se lo llama mutables, mientras que si el objeto no puede cambiar, el objeto es inmutable.

### type()

La función type() devuelve el tipo del objeto.

### id()

La función id() recibe un objeto como argumento y nos devuelve un valor que representa la dirección de memoria donde está almacenado el objeto. Se lo utiliza como “identificador” único para el objeto enviado por argumento.

## Objetos inmutables

Significa que estos objetos no pueden modificarse una vez creados. Son:

- los números
- las cadenas de texto
- las tuplas y frozensets

Veamos un ejemplo con cadenas de texto. Imaginemos que necesitamos cambiar una letra de un string.

```
nombre = 'Gerarda'
print(nombre[6]) # a
nombre[6] = 'o'
# TypeError: 'str' object does not support item assignment
```

Podemos tratar de resolverlo asignando un nuevo valor a la variable, pero si revisamos los id antes y después, podemos confirmar que el objeto original no fue modificado, sino que se creó uno nuevo en otro lugar de la memoria. La cadena no cambió. Se “guardó” un **objeto diferente** dentro de la variable.

```
print(id(nombre)) # 140640984908272
nombre = 'Gerardo'
print(id(nombre)) # 140640984908208
```

## Objetos mutables

Los valores de los objetos mutables se pueden cambiar en cualquier momento y en cualquier lugar después de su creación. Son:

- las listas
- los sets
- los diccionarios

Veamos un ejemplo con listas a partir del anterior:

```
nombre = ['G', 'e', 'r', 'a', 'r', 'd', 'a']
print(nombre) # ['G', 'e', 'r', 'a', 'r', 'd', 'a']
print(nombre[6]) # a
print(id(nombre)) # 140171651834624
nombre[6] = 'o'
print(nombre) # ['G', 'e', 'r', 'a', 'r', 'd', 'o']
print(id(nombre)) # 140171651834624
```

Se puede observar que no sólo podemos cambiar una parte de la lista, sino que podemos comprobar que se trata del mismo objeto antes y después del cambio.

## Implicancias

- Los objetos inmutables con más rápidos de acceder que los inmutables.
- Cambiar un objeto inmutable implica crear una copia (caro en recursos)
- Los objetos mutables son ideales para datos que se modifican dinámicamente.



## Argumentos de funciones inmutables

```
def incrementa(nro):  
    print(id(nro)) # 9789248  
    nro += 1  
    print(nro) # 11  
    print(id(nro)) # 9789280  
  
num = 10  
print(id(num)) # 9789248  
incrementa(num)  
print(num) # 10  
print(id(num)) # 9789248
```

El número (objeto inmutable), no puede ser modificado. Dentro de la función, se crea una copia para poder aplicar la modificación.

**Una vez fuera de la función, el número conserva su valor original.**

## Argumentos de funciones mutables

En este caso, utilizamos una lista de un solo objeto. Lo enviamos a la función, que lo modifica y lo devuelve al programa principal. Se trabaja siempre sobre el mismo objeto.

**Una vez fuera de la función, la lista sigue con su valor modificado.**

```
def incrementa(nro):  
    print(id(nro)) # 139830809498560  
    nro += [156]  
    print(nro) # [10, 156]  
    print(id(nro)) # 139830809498560  
  
num = [10]  
print(id(num)) # 139830809498560  
incrementa(num)  
print(num) # [10, 156]  
print(id(num)) # 139830809498560
```