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.
- f**ronzenset([iterable]):** devuelve un objeto frozenset, opcionalmente con elementos del iterable.
- **help([objeto]): i**nvoca 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]): r**etorna 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): r**etorna 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 para 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

Métodos de las cadenas (str)

.1 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.

.2 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.

.3 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.

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.truc(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, π]
- **math.asin(x):** retorna el arcoseno de x, en radianes. [-π/2, π/2]
- **math.atan(x):** retorna la arcotangente de x en radianes. [-π/2, π/2]
- **math.atan2(x, y):** retorna atan(y / x), en radianes [-π, π]
- **math.cos(x):** retorna el coseno de x en radianes.
- **math.sin(x):** retorna el seno de x en radianes.
- **math.tan(x):** retona 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.

- ⑩ **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

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.

```python
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)
```

```python
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.

```python
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.

```python
import random as r

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

'''
0.8992764679053034
0.0740968548328993
0.8180031166838229
0.40850380396017527
0.6422579590270423
'''
```

- **random.uniform():** devuelve un número float incluido entre los valores indicados.
- **random.seed(): s**e 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))
```

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
```

```
# 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
```

```
# 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
```

```python
# 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'''
```

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
```

1 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
```

2 Creando fechas

```
import datetime

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

3 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:

```python
import datetime

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

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.

1 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 cambar 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
```

2 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.

```python
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
```