Funciones Incorporadas Basicas para empezar en python

A	B	C	D
abs()	bin()	callable()	delattr()
aiter()	bool()	chr()	dict()
all()	breakpoint()	classmethod()	dir()
anext()	bytearray()	compile()	divmod()
any()	bytes()	complex()	
ascii()			

E	F	G	H
enumerate()	filter()	getattr()	hasattr()
eval()	float()	globals()	hash()
exec()	format()		help()
	frozenset()		hex()

I	L	M	N
id()	len()	map()	next()
input()	list()	max()	
int()	locals()	memoryview()	
isinstance()		min()	
issubclass()			

O	P	R	S
object()	pow()	range()	set()
oct()	print()	repr()	setattr()
open()	property()	reversed()	slice()
ord()		round()	sorted()
			staticmethod()
			str()
			sum()
			super()

T	V	Z	_
tuple()	vars()	zip()	**import**()
type()			

Por supuesto, puedo convertir la lista que proporcionaste a Markdown y darte un ejemplo de cada uno de los términos, también en Markdown. Aquí tienes:

A

- **abs()** : Retorna el valor absoluto de un número.

```
print(abs(-5))  # Salida: 5
```

- **aiter()** : Retorna un objeto iterador para un objeto iterable.

```
# Ejemplo pendiente hasta que la función esté disponible en Python.
```

- **all()** : Retorna True si todos los elementos de un iterable son verdaderos.

```
print(all([True, True, True]))  # Salida: True
```

- **anext()** : Devuelve el siguiente elemento de un iterador.

```
# Ejemplo pendiente hasta que la función esté disponible en Python.
```

- **any()** : Retorna True si algún elemento de un iterable es verdadero.

```
print(any([False, True, False]))  # Salida: True
```

- **ascii()** : Retorna una representación imprimible de un objeto, escapando los caracteres no-ASCII.

```
print(ascii('ñandú'))  # Salida: '\\xf1and\\xfa'
```

B

- **bin()** : Convierte un número entero en su representación binaria.

```
print(bin(10))  # Salida: '0b1010'
```

- **bool()** : Retorna el valor booleano de un objeto.

```
print(bool(0))  # Salida: False
```

C

- **callable()** : Retorna True si el objeto especificado es "llamable".

```
print(callable(print))  # Salida: True
```

- **chr()** : Retorna un carácter representado por un número entero.

```
print(chr(97))  # Salida: 'a'
```

- **classmethod()** : Retorna un método de clase para una función.

```
class MiClase:
    @classmethod
    def decir_hola(cls):
        return '¡Hola!'
print(MiClase.decir_hola())  # Salida: '¡Hola!'
```

- **`compile()`** : Compila el código fuente en un objeto código o AST.

```
codigo = 'a = 5\nb=10\nprint("Suma:", a+b)'
codigo_compilado = compile(codigo, 'sumador', 'exec')
exec(codigo_compilado)
```

- **`complex()`** : Retorna un número complejo con los valores especificados.

```
print(complex(2, 3))  # Salida: (2+3j)
```

D

- **`delattr()`** : Elimina el atributo especificado de un objeto.

```
class Objeto:
    atributo = 42
delattr(Objeto, 'atributo')
```

- **`dict()`** : Retorna un diccionario construido a partir de argumentos de palabra clave o un objeto iterable de pares clave-valor.

```
print(dict(a=1, b=2))  # Salida: {'a': 1, 'b': 2}
```

- **`divmod()`** : Retorna el cociente y el resto de la división de dos números.

```
print(divmod(7, 2))  # Salida: (3, 1)
```

- **`dir()`** : Retorna una lista de nombres de los atributos y métodos de un objeto.

```
print(dir([]))  # Salida: ['__add__', '__class__', ...]
```

E

- **`enumerate()`** : Retorna un objeto enumerado.

```
for indice, valor in enumerate(['a', 'b', 'c']):
    print(indice, valor)  # Salida: 0 a, 1 b, 2 c
```

F

- **`filter()`** : Construye un iterador a partir de elementos de un iterable para los cuales una función retorna True.

```
numeros = [1, 2, 3, 4, 5]
pares = filter(lambda x: x % 2 == 0, numeros)
print(list(pares))  # Salida: [2, 4]
```

G

- **`getattr()`** : Retorna el valor del atributo especificado de un objeto.

```
class Objeto:
    atributo = 'valor'
print(getattr(Objeto, 'atributo'))  # Salida: 'valor'
```

- **`globals()`** : Retorna el diccionario global actual de símbolos.

```
print(globals())  # Salida: {...}
```

H

- **`hash()`** : Retorna el valor hash de un objeto.

```
print(hash('test'))  # Salida: 2314058222102390712 (el valor puede variar)
```

- **`help()`** : Abre el sistema de ayuda interactivo de Python.

```
# Ejemplo no aplicable en Markdown.
```

I

- **`id()`** : Retorna la identificación única de un objeto.

```
objeto = 'hola'
print(id(objeto))  # Salida: 139954740223168 (el valor puede variar)
```

- **`input()`** : Lee una entrada del usuario desde la consola.

```
# Ejemplo no aplicable en Markdown.
```

- **`int()`** : Retorna un número entero construido a partir de un número o una cadena.

```
print(int('10'))  # Salida: 10
```

- **`isinstance()`** : Retorna True si un objeto es una instancia de una clase o de cualquier clase derivada de ella.

```
print(isinstance(5, int))  # Salida: True
```

- **`issubclass()`** : Retorna True si una clase es una subclase de otra.

```
class Padre:
    pass
class Hijo(Padre):
    pass
print(issubclass(Hijo, Padre))  # Salida: True
```

L

- **`len()`** : Retorna la longitud de un objeto.

```
print(len('hola'))  # Salida: 4
```

- **`list()`** : Retorna una lista construida a partir de un iterable.

```
print(list('abc'))  # Salida: ['a', 'b', 'c']
```

- **`locals()`** : Retorna el diccionario local actual de símbolos.

```
def funcion():
    local = 5
    print(locals())
funcion()  # Salida: {'local': 5}
```

M

- **`map()`** : Aplica una función a todos los elementos de un iterable.

```
numeros = [1, 2, 3, 4, 5]
cuadrados = map(lambda x: x**2, numeros)
print(list(cuadrados))  # Salida: [1, 4, 9, 16, 25]
```

- **`max()`** : Retorna el elemento más grande de un iterable.

```
print(max([1, 2, 3]))  # Salida: 3
```

- **`memoryview()`** : Retorna un objeto de vista de memoria de un objeto.

```
bytes_obj = bytes('hola', 'utf-8')
vista_memoria = memoryview(bytes_obj)
print(vista_memoria[0])  # Salida: 104
```

- **`min()`** : Retorna el elemento más pequeño de un iterable.

```
print(min([1, 2, 3]))  # Salida: 1
```

N

- **`next()`** : Retorna el siguiente elemento de un iterador.

```
iterador = iter([1, 2, 3])
print(next(iterador))  # Salida: 1
```

O

- **`object()`** : Retorna un nuevo objeto.

```
nuevo_objeto = object()
print(nuevo_objeto)  # Salida: <object object at 0x7f4e6c7d1b70>
```

- **`oct()`** : Convierte un número entero en su representación octal.

```
print(oct(8))  # Salida: '0o10'
```

P

- **`pow()`** : Retorna el resultado de elevar un número a una potencia.

```
print(pow(2, 3))  # Salida: 8
```

- **`print()`**: Imprime un mensaje en la consola.

```
print('Hola mundo')  # Salida: Hola mundo
```

- **`property()`**: Retorna una propiedad.

```
class MiClase:
    def __
```

Por supuesto, aquí tienes los ejemplos en Markdown para los términos que mencionaste:

R

- **`range()`**: Retorna una secuencia inmutable de números.

```
for i in range(5):
    print(i)  # Salida: 0 1 2 3 4
```

- **`repr()`**: Retorna una representación imprimible de un objeto.

```
objeto = 'Hola'
print(repr(objeto))  # Salida: 'Hola'
```

- **`reversed()`**: Retorna un iterador que recorre los elementos de un iterable en orden inverso.

```
for i in reversed([1, 2, 3]):
    print(i)  # Salida: 3 2 1
```

- **`round()`**: Redondea un número al entero más cercano.

```
print(round(3.14159))  # Salida: 3
```

S

- **`setattr()`**: Establece el valor de un atributo de un objeto.

```
class Objeto:
    pass
objeto = Objeto()
setattr(objeto, 'atributo', 100)
print(objeto.atributo)  # Salida: 100
```

- **`slice()`**: Retorna un objeto slice que puede ser utilizado para hacer slicing en secuencias.

```
lista = [0, 1, 2, 3, 4]
corte = slice(1, 3)
print(lista[corte])  # Salida: [1, 2]
```

- **`sorted()`**: Retorna una lista ordenada de los elementos de un iterable.

```
desordenado = [3, 1, 2]
print(sorted(desordenado))  # Salida: [1, 2, 3]
```

- **`staticmethod()`**: Retorna un método estático para una función.

```
class MiClase:
    @staticmethod
    def ejemplo():
        return 'Estático'
print(MiClase.ejemplo())  # Salida: 'Estático'
```

- **`str()`**: Retorna una representación de cadena de un objeto.

```
numero = 123
print(str(numero))  # Salida: '123'
```

- **`sum()`**: Retorna la suma de todos los elementos de un iterable.

```
numeros = [1, 2, 3]
print(sum(numeros))  # Salida: 6
```

- **`super()`**: Retorna un objeto proxy que delega las llamadas a un objeto padre o hermano.

```
class Padre:
    def decir(self):
        return 'Hola'
class Hijo(Padre):
    def decir(self):
        return super().decir() + ', mundo'
hijo = Hijo()
print(hijo.decir())  # Salida: 'Hola, mundo'
```

T

- **`tuple()`**: Retorna una tupla construida a partir de un iterable.

```
print(tuple([1, 2, 3]))  # Salida: (1, 2, 3)
```

V

- **`vars()`**: Retorna el diccionario **`dict`** de un objeto.

```
class Objeto:
    def __init__(self):
        self.atributo = 'valor'
objeto = Objeto()
print(vars(objeto))  # Salida: {'atributo': 'valor'}
```

Z

- **`zip()`**: Retorna un iterador de tuplas, donde la i-ésima tupla contiene el i-ésimo elemento de cada uno de los argumentos.

```
a = [1, 2, 3]
b = ['a', 'b', 'c']
```

```
for i in zip(a, b):
    print(i)  # Salida: (1, 'a') (2, 'b') (3, 'c')
```

—

- `__import__()`: Importa un módulo de forma dinámica.

```
json = __import__('json')
print(json.dumps({'Python': 3.8}))  # Salida: '{"Python": 3.8}'
```