1. print(): Imprime objetos en la consola. Básico:

python

```python
print("Hola, mundo!")  # Hola, mundo!
```

Avanzado:

python

```python
import sys
print("Error:", "Archivo no encontrado", file=sys.stderr, sep=" | ", end="\n\n")
```

2. len(): Devuelve la longitud de un objeto. Básico:

python

```python
print(len("Python"))  # 6
```

Avanzado:

python

```python
class MiObjeto:
    def __len__(self):
        return 42

obj = MiObjeto()
print(len(obj))  # 42
```

3. type(): Devuelve el tipo de un objeto. Básico:

python

```python
print(type(42))  # <class 'int'>
```

Avanzado:

python

```python
def crear_clase(nombre):
    return type(nombre, (), {"saludo": lambda self: f"Hola, soy {nombre}"})

MiClase = crear_clase("MiClase")
obj = MiClase()
print(obj.saludo())  # Hola, soy MiClase
```

4. int(): Convierte un objeto a entero. Básico:

python

```python
print(int("42"))  # 42
```

Avanzado:

python

```python
class ConvertibleAEntero:
    def __int__(self):
        return 42

obj = ConvertibleAEntero()
print(int(obj))  # 42
```

5. float(): Convierte un objeto a número de punto flotante. Básico:

python

```python
print(float("3.14"))  # 3.14
```

Avanzado:

python

```python
import struct

def float_a_bytes(f):
```

```python
    return struct.pack('>f', f)

def bytes_a_float(b):
    return struct.unpack('>f', b)[0]

original = 3.14
como_bytes = float_a_bytes(original)
reconstruido = bytes_a_float(como_bytes)
print(reconstruido)  # 3.14
```

6. str(): Convierte un objeto a cadena. Básico:

python

```python
print(str(42))  # "42"
```

Avanzado:

python

```python
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def __str__(self):
        return f"{self.nombre} ({self.edad} años)"

p = Persona("Ana", 30)
print(str(p))  # Ana (30 años)
```

7. list(): Crea una lista o convierte un iterable a lista. Básico:

python

```python
print(list("Python"))  # ['P', 'y', 't', 'h', 'o', 'n']
```

Avanzado:

python

```python
def fibonacci_generator():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b

fib = fibonacci_generator()
fib_list = list(itertools.islice(fib, 10))
print(fib_list)  # [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

8. tuple(): Crea una tupla o convierte un iterable a tupla. Básico:

python

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

Avanzado:

python

```python
from collections import namedtuple

Punto = namedtuple('Punto', ['x', 'y'])
p = Punto(1, 2)
print(p.x, p.y)  # 1 2
```

9. set(): Crea un conjunto o convierte un iterable a conjunto. Básico:

python

```python
print(set([1, 2, 2, 3]))  # {1, 2, 3}
```

Avanzado:

python

```python
def encontrar_caracteres_unicos(texto):
    return set(texto.lower()) - set(' ,.!?')

print(encontrar_caracteres_unicos("Hola, Mundo!"))  # {'h', 'o', 'l', 'a', 'm', 'u', 'n', 'd'}
```

10. dict(): Crea un diccionario o convierte pares clave-valor a diccionario. Básico:

python

```python
print(dict([(1, 'uno'), (2, 'dos')]))  # {1: 'uno', 2: 'dos'}
```

Avanzado:

python

```python
from collections import defaultdict

texto = "el gato en el tejado"
conteo = defaultdict(int)
for palabra in texto.split():
    conteo[palabra] += 1
print(dict(conteo))  # {'el': 2, 'gato': 1, 'en': 1, 'tejado': 1}
```

11. range(): Genera una secuencia de números. Básico:

python

```python
print(list(range(5)))  # [0, 1, 2, 3, 4]
```

Avanzado:

python

```python
def es_primo(n):
    return n > 1 and all(n % i != 0 for i in range(2, int(n**0.5) + 1))
```

```python
primos = [num for num in range(2, 100) if es_primo(num)]
print(primos)  # [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```

12. sum(): Suma los elementos de un iterable. Básico:

python

```python
print(sum([1, 2, 3, 4, 5]))  # 15
```

Avanzado:

python

```python
from functools import reduce
import operator

def suma_matrices(matrices):
    return reduce(lambda x, y: [[sum(valores) for valores in zip(*filas)] for filas in zip(x, y)], matrices)

m1 = [[1, 2], [3, 4]]
m2 = [[5, 6], [7, 8]]
m3 = [[9, 10], [11, 12]]
print(suma_matrices([m1, m2, m3]))  # [[15, 18], [21, 24]]
```

13. max(): Devuelve el elemento más grande de un iterable o el mayor de dos o más argumentos. Básico:

python

```python
print(max([1, 5, 3, 2]))  # 5
```

Avanzado:

python

```python
estudiantes = [
    {'nombre': 'Ana', 'nota': 95},
    {'nombre': 'Bob', 'nota': 75},
    {'nombre': 'Charlie', 'nota': 85}
```

```python
]
mejor_estudiante = max(estudiantes, key=lambda x: x['nota'])
print(mejor_estudiante['nombre'])  # Ana
```

14. min(): Devuelve el elemento más pequeño de un iterable o el menor de dos o más argumentos. Básico:

```python
print(min([1, 5, 3, 2]))  # 1
```

Avanzado:

```python
import heapq

def encontrar_k_menores(numeros, k):
    return heapq.nsmallest(k, numeros)

print(encontrar_k_menores([5, 2, 9, 1, 7, 6, 3], 3))  # [1, 2, 3]
```

15. abs(): Devuelve el valor absoluto de un número. Básico:

```python
print(abs(-42))  # 42
```

Avanzado:

```python
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __abs__(self):
        return (self.x**2 + self.y**2)**0.5
```

```python
v = Vector(3, 4)
print(abs(v))  # 5.0
```

16. round(): Redondea un número a un número dado de decimales. Básico:

python

```python
print(round(3.14159, 2))  # 3.14
```

Avanzado:

python

```python
def redondear_a_multiplo(numero, multiplo):
    return round(numero / multiplo) * multiplo

print(redondear_a_multiplo(42, 5))  # 40
print(redondear_a_multiplo(44, 5))  # 45
```

17. sorted(): Devuelve una nueva lista ordenada de un iterable. Básico:

python

```python
print(sorted([3, 1, 4, 1, 5, 9, 2]))  # [1, 1, 2, 3, 4, 5, 9]
```

Avanzado:

python

```python
from operator import itemgetter

empleados = [
    {'nombre': 'Ana', 'salario': 50000, 'antiguedad': 5},
    {'nombre': 'Bob', 'salario': 60000, 'antiguedad': 3},
    {'nombre': 'Charlie', 'salario': 55000, 'antiguedad': 7}
]
```

```python
print(sorted(empleados, key=itemgetter('salario', 'antiguedad'), reverse=True))
```

18. reversed(): Devuelve un iterador que accede a la secuencia en orden inverso. Básico:

python

```python
print(list(reversed([1, 2, 3])))  # [3, 2, 1]
```

Avanzado:

python

```python
class CuentaRegresiva:
    def __init__(self, start):
        self.start = start

    def __iter__(self):
        return self

    def __next__(self):
        if self.start <= 0:
            raise StopIteration
        self.start -= 1
        return self.start + 1

    def __reversed__(self):
        return range(1, self.start + 1)

for i in reversed(CuentaRegresiva(5)):
    print(i, end=' ')  # 1 2 3 4 5
```

19. enumerate(): Devuelve un objeto enumerate (índice, valor). Básico:

python

```python
print(list(enumerate(['a', 'b', 'c'])))  # [(0, 'a'), (1, 'b'), (2, 'c')]
```

Avanzado:

```python
def encontrar_indices(lista, valor):
    return [indice for indice, elemento in enumerate(lista) if elemento == valor]

numeros = [1, 2, 3, 2, 4, 2, 5]
print(encontrar_indices(numeros, 2))  # [1, 3, 5]
```

20. zip(): Combina elementos de dos o más iterables. Básico:

```python
print(list(zip([1, 2, 3], ['a', 'b', 'c'])))  # [(1, 'a'), (2, 'b'), (3, 'c')]
```

Avanzado:

```python
def transponer_matriz(matriz):
    return list(map(list, zip(*matriz)))

matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(transponer_matriz(matriz))  # [[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

21. any(): Devuelve True si algún elemento del iterable es True. Básico:

```python
print(any([False, True, False]))  # True
```

Avanzado:

```python
def tiene_duplicados(iterable):
    seen = set()
    return any(i in seen or seen.add(i) for i in iterable)
```

```python
print(tiene_duplicados([1, 2, 3, 4, 5]))  # False
print(tiene_duplicados([1, 2, 3, 2, 4]))  # True
```

22. all(): Devuelve True si todos los elementos del iterable son True. Básico:

python

```python
print(all([True, True, True]))  # True
```

Avanzado:

python

```python
def es_sudoku_valido(tablero):
    filas = tablero
    columnas = zip(*tablero)
    cuadrados = [
        [tablero[i+m][j+n] for m in range(3) for n in range(3)]
        for i in range(0, 9, 3) for j in range(0, 9, 3)
    ]

    return all(
        set(grupo) == set(range(1, 10))
        for grupo in filas + list(columnas) + cuadrados
    )

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

```
]
print(es_sudoku_valido(sudoku))  # True
```