Resumen Clave de Python

Basado en el Tutorial Oficial de Python (docs.python.org)

Academia Bio-Python

Guia completa de referencia para aprender a programar en Python

Febrero 2026

Índice

1. Numeros, Texto y Listas

1.1 Numeros

1.1.1 Tipos numericos

Tipo	Descripcion	Ejemplo
`int`	Numeros enteros	`2, 42, -7`
`float`	Numeros con decimales	`3.14, -0.5`
`complex`	Numeros complejos	`3+5j`
`Decimal`	Aritmetica decimal precisa	`Decimal('0.1')`
`Fraction`	Numeros racionales	`Fraction(1, 3)`

1.1.2 Operadores aritmeticos

```
2 + 3        # 5    Suma
10 - 4       # 6    Resta
3 * 7        # 21   Multiplicacion
8 / 5        # 1.6  Division (siempre devuelve float)
17 // 3      # 5    Division entera (descarta decimales)
17 % 3       # 2    Modulo (resto de la division)
5 ** 2       # 25   Potencia
```

1.1.3 Asignacion de variables

```
width = 20
height = 5 * 9
area = width * height     # 900

# Asignacion multiple
a, b = 0, 1

# Variable especial _ (ultimo resultado en modo interactivo)
>>> 100.50 * 0.125
12.5625
>>> _ + 100
112.5625
```

1.2 Texto (Strings)

1.2.1 Creacion de cadenas

```
# Comillas simples o dobles
'hola mundo'
"hola mundo"

# Escapar comillas
'doesn\'t'
"doesn't"

# Cadenas multilinea (triple comillas)
texto = """Primera linea
Segunda linea
Tercera linea"""

# Cadenas crudas (raw) - ignoran secuencias de escape
```

```
print(r'C:\nueva\carpeta')    # C:\nueva\carpeta
```

1.2.2 Indexacion y rebanado (slicing)

```
word = 'Python'
#  P  y  t  h  o  n
#  0  1  2  3  4  5   (indices positivos)
# -6 -5 -4 -3 -2 -1   (indices negativos)

word[0]       # 'P'    primer caracter
word[-1]      # 'n'    ultimo caracter
word[0:2]     # 'Py'   del 0 al 2 (excluido)
word[2:]      # 'thon' del 2 al final
word[:4]      # 'Pyth' del inicio al 4 (excluido)
word[-2:]     # 'on'   ultimos 2 caracteres
```

1.2.3 Operaciones con cadenas

```
# Concatenacion
'Py' + 'thon'          # 'Python'
3 * 'un' + 'ium'       # 'unununium'

# Literales adyacentes se concatenan automaticamente
'Py' 'thon'            # 'Python'

# Las cadenas son INMUTABLES
word[0] = 'J'   # ERROR: TypeError

# Para modificar, crear nueva cadena
'J' + word[1:]  # 'Jython'

# Longitud
len('Python')  # 6
```

1.3 Listas

1.3.1 Creacion y operaciones basicas

```
squares = [1, 4, 9, 16, 25]

# Indexacion y slicing (igual que strings)
squares[0]        # 1
squares[-1]       # 25
squares[1:3]      # [4, 9]

# Las listas son MUTABLES
squares[3] = 64  # [1, 4, 9, 64, 25]

# Concatenacion
squares + [36, 49]  # [1, 4, 9, 64, 25, 36, 49]

# Agregar elementos
squares.append(36)

# Longitud
len(squares)      # 6
```

```
# Listas anidadas
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
matrix[0][1]      # 2
```

1.3.2 Asignacion por rebanado

```
letters = ['a', 'b', 'c', 'd', 'e']
letters[2:5] = ['C', 'D', 'E']    # reemplazar
letters[2:5] = []                   # eliminar
letters[:] = []                     # limpiar toda la lista
```

Cuidado con referencias: `rgba = rgb` NO copia la lista, ambas apuntan al mismo objeto. Para copiar: `copia = original[:]` o `copia = original.copy()`

2. Control de Flujo

2.1 Sentencia if / elif / else

```
x = int(input("Ingrese un numero: "))
if x < 0:
    print('Negativo')
elif x == 0:
    print('Cero')
elif x == 1:
    print('Uno')
else:
    print('Mayor que uno')
```

`elif` es abreviatura de "else if". Puede haber cero o mas bloques `elif`. El bloque `else` es opcional.

2.2 Bucle while

```
a, b = 0, 1
while a < 1000:
    print(a, end=',')
    a, b = b, a + b
# Salida: 0,1,1,2,3,5,8,13,21,34,55,89,...
```

2.3 Bucle for

```
# Iterar sobre una secuencia
words = ['gato', 'ventana', 'defenestrar']
for w in words:
    print(w, len(w))

# NUNCA modificar una coleccion mientras se itera
# Solucion 1: iterar sobre una copia
for user, status in users.copy().items():
    if status == 'inactive':
        del users[user]

# Solucion 2: crear nueva coleccion
active = {u: s for u, s in users.items() if s == 'active'}
```

2.4 La funcion range()

```
range(5)             # 0, 1, 2, 3, 4
range(5, 10)         # 5, 6, 7, 8, 9
range(0, 10, 3)      # 0, 3, 6, 9
range(-10, -100, -30)  # -10, -40, -70

# Iterar sobre indices de una secuencia
a = ['Mary', 'had', 'a', 'little', 'lamb']
for i in range(len(a)):
    print(i, a[i])

# Mejor alternativa: enumerate()
for i, item in enumerate(a):
    print(i, item)
```

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

2.5 break, continue y else en bucles

```python
# break: sale del bucle inmediatamente
# continue: pasa a la siguiente iteracion
# else en bucle: se ejecuta si NO se uso break

for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print(f"{n} = {x} * {n//x}")
            break
    else:
        # Se ejecuta si el for termina sin break
        print(f"{n} es primo")
```

2.6 Sentencia pass

```python
# No hace nada, sirve como placeholder
class MiClaseVacia:
    pass

def funcion_pendiente():
    pass   # TODO: implementar
```

2.7 Sentencia match (Python 3.10+)

```python
def http_error(status):
    match status:
        case 400:
            return "Bad request"
        case 404:
            return "Not found"
        case 401 | 403:
            return "Not allowed"
        case _:             # _ es comodin
            return "Error desconocido"

# Con tuplas y variables
match punto:
    case (0, 0):
        print("Origen")
    case (0, y):
        print(f"Y={y}")
    case (x, 0):
        print(f"X={x}")
    case (x, y):
        print(f"X={x}, Y={y}")

# Con guardas (if)
match punto:
    case Point(x, y) if x == y:
        print(f"En diagonal en {x}")
```

3. Funciones

3.1 Definicion basica

```
def fibonacci(n):
    """Imprime la serie de Fibonacci hasta n."""  # docstring
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a + b
    print()

fibonacci(2000)

# Las funciones sin return devuelven None
# Las funciones son objetos de primera clase
f = fibonacci
f(100)
```

3.2 Funciones con retorno

```
def fib_lista(n):
    """Retorna una lista con la serie Fibonacci hasta n."""
    resultado = []
    a, b = 0, 1
    while a < n:
        resultado.append(a)
        a, b = b, a + b
    return resultado

nums = fib_lista(100)  # [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

3.3 Argumentos por defecto

```
def preguntar(prompt, reintentos=4, mensaje='Intente de nuevo'):
    while True:
        respuesta = input(prompt)
        if respuesta in {'s', 'si'}:
            return True
        if respuesta in {'n', 'no'}:
            return False
        reintentos -= 1
        if reintentos < 0:
            raise ValueError('Respuesta invalida')
        print(mensaje)
```

CUIDADO: Los valores por defecto se evaluan UNA sola vez (al definir la funcion). Nunca usar mutables como valor por defecto:

```
# MAL - la lista se comparte entre llamadas
def f(a, L=[]):
    L.append(a)
    return L

# BIEN - usar None como centinela
def f(a, L=None):
```

```
    if L is None:
        L = []
    L.append(a)
    return L
```

3.4 Argumentos por palabra clave

```
def describir(voltaje, estado='rigido', accion='voom'):
    print(f"Voltaje: {voltaje}, Estado: {estado}, Accion: {accion}")

# Llamadas validas
describir(1000)
describir(voltage=1000)
describir(voltaje=1000, accion='BOOM')
describir('un millon', estado='inerte')

# Llamadas INVALIDAS
describir()                      # falta argumento requerido
describir(voltaje=5.0, 'muerto') # posicional despues de keyword
describir(110, voltaje=220)      # duplicado
```

3.5 Parametros especiales: /, *

```
def f(solo_pos, /, pos_o_kwd, *, solo_kwd):
    pass
#     ----------   ----------   ----------
#     Solo          Posicional   Solo
#     posicional    o keyword    keyword

# Ejemplo
def f(a, b, /, c, d, *, e, f):
    print(a, b, c, d, e, f)

f(1, 2, 3, d=4, e=5, f=6)        # OK
f(1, 2, c=3, d=4, e=5, f=6)      # OK
f(a=1, b=2, c=3, d=4, e=5, f=6) # ERROR: a,b son solo posicionales
```

3.6 *args y **kwargs

```
# *args recoge argumentos posicionales extras en una tupla
# **kwargs recoge argumentos keyword extras en un diccionario

def funcion(*args, **kwargs):
    print("Posicionales:", args)
    print("Keywords:", kwargs)

funcion(1, 2, 3, nombre="Ana", edad=25)
# Posicionales: (1, 2, 3)
# Keywords: {'nombre': 'Ana', 'edad': 25}

# Desempaquetado
args = [3, 6]
list(range(*args))    # [3, 4, 5]

d = {"voltaje": "1M", "estado": "muerto"}
describir(**d)
```

3.7 Expresiones lambda

```
# Funciones anonimas de una sola expresion
cuadrado = lambda x: x ** 2
suma = lambda a, b: a + b

# Uso comun: como argumento de funciones
pares = [(1, 'uno'), (2, 'dos'), (3, 'tres')]
pares.sort(key=lambda par: par[1])  # ordena por segundo elemento

# Como fabrica de funciones
def incrementador(n):
    return lambda x: x + n
f = incrementador(42)
f(1)  # 43
```

3.8 Docstrings y anotaciones

```
def mi_funcion(nombre: str, edad: int = 0) -> str:
    """Retorna un saludo personalizado.

    Argumentos:
        nombre: El nombre de la persona.
        edad: La edad (opcional).
    """
    return f"Hola {nombre}, tienes {edad} anios"

# Acceso al docstring
print(mi_funcion.__doc__)

# Acceso a las anotaciones
print(mi_funcion.__annotations__)
# {'nombre': str, 'edad': int, 'return': str}
```

4. Estructuras de Datos

4.1 Metodos de listas

Metodo	Descripcion
`list.append(x)`	Agrega un elemento al final
`list.extend(iterable)`	Extiende con elementos de un iterable
`list.insert(i, x)`	Inserta en la posicion `i`
`list.remove(x)`	Elimina primera ocurrencia de `x`
`list.pop([i])`	Elimina y retorna el elemento en `i` (ultimo si no se da `i`)
`list.clear()`	Elimina todos los elementos
`list.index(x)`	Indice de la primera ocurrencia de `x`
`list.count(x)`	Cuenta las ocurrencias de `x`
`list.sort()`	Ordena la lista in-place
`list.reverse()`	Invierte la lista in-place
`list.copy()`	Retorna una copia superficial

4.1.1 Listas como pilas (LIFO)

```
pila = [3, 4, 5]
pila.append(6)      # apilar
pila.append(7)
pila.pop()          # desapilar -> 7
pila.pop()          # -> 6
```

4.1.2 Listas como colas (FIFO) - usar deque

```
from collections import deque
cola = deque(["Eric", "John", "Michael"])
cola.append("Terry")        # llega Terry
cola.popleft()               # sale Eric
cola.popleft()               # sale John
# deque(['Michael', 'Terry'])
```

4.2 Comprension de listas

```
# Sintaxis: [expresion for item in iterable if condicion]

cuadrados = [x**2 for x in range(10)]
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

# Filtrar
pares = [x for x in range(20) if x % 2 == 0]

# Combinar dos secuencias
combos = [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]

# Aplicar funciones
limpio = [s.strip() for s in ['  banana', ' fresa  ']]

# Aplanar lista de listas
flat = [n for fila in [[1,2],[3,4],[5,6]] for n in fila]
# [1, 2, 3, 4, 5, 6]
```

```
# Comprension anidada (transponer matriz)
matrix = [[1,2,3], [4,5,6], [7,8,9]]
transpuesta = [[fila[i] for fila in matrix] for i in range(3)]
# Mejor alternativa: list(zip(*matrix))
```

4.3 La sentencia del

```
a = [1, 2, 3, 4, 5]
del a[0]        # elimina por indice
del a[1:3]      # elimina rebanado
del a[:]        # limpia la lista
del a           # elimina la variable
```

4.4 Tuplas

```
# Inmutables, pueden contener elementos heterogeneos
t = 12345, 54321, 'hola'
t[0]          # 12345
# t[0] = 0  # ERROR: las tuplas son inmutables

# Pero pueden contener objetos mutables
v = ([1, 2], [3, 4])

# Tupla vacia y de un elemento
vacia = ()
singleton = ('hola',)    # nota la coma final

# Empaquetado y desempaquetado
t = 1, 2, 3             # empaquetado
x, y, z = t             # desempaquetado
```

4.5 Conjuntos (Sets)

```
# Coleccion sin duplicados, no ordenada
canasta = {'manzana', 'naranja', 'manzana', 'pera'}
# {'manzana', 'naranja', 'pera'}

# Conjunto vacio: set() (NO {}, eso es un dict vacio)
vacio = set()

# Pertenencia
'naranja' in canasta    # True

# Operaciones matematicas
a = set('abracadabra')   # {'a', 'b', 'c', 'd', 'r'}
b = set('alacazam')      # {'a', 'c', 'l', 'm', 'z'}

a - b    # Diferencia:    {'b', 'd', 'r'}
a | b    # Union:         {'a', 'b', 'c', 'd', 'l', 'm', 'r', 'z'}
a & b    # Interseccion:  {'a', 'c'}
a ^ b    # Diferencia simetrica: {'b', 'd', 'l', 'm', 'r', 'z'}

# Comprension de conjuntos
{x for x in 'abracadabra' if x not in 'abc'}  # {'d', 'r'}
```

4.6 Diccionarios

```
# Pares clave:valor. Claves deben ser inmutables.
tel = {'jack': 4098, 'sape': 4139}

tel['guido'] = 4127        # agregar
tel['jack']                # acceder -> 4098
del tel['sape']            # eliminar
list(tel)                  # lista de claves
sorted(tel)                # claves ordenadas
'guido' in tel             # True

# Crear desde secuencia de pares
dict([('a', 1), ('b', 2)])

# Comprension de diccionarios
{x: x**2 for x in (2, 4, 6)}  # {2: 4, 4: 16, 6: 36}

# Con argumentos keyword
dict(uno=1, dos=2, tres=3)
```

4.7 Tecnicas de iteracion

```
# items() - clave y valor de diccionarios
for k, v in diccionario.items():
    print(k, v)

# enumerate() - indice y valor
for i, v in enumerate(['a', 'b', 'c']):
    print(i, v)   # 0 a, 1 b, 2 c

# zip() - emparejar secuencias
preguntas = ['nombre', 'color']
respuestas = ['Lancelot', 'azul']
for p, r in zip(preguntas, respuestas):
    print(f'{p}: {r}')

# reversed() - iterar en reversa
for i in reversed(range(1, 10, 2)):
    print(i)   # 9, 7, 5, 3, 1

# sorted() - iterar ordenado (no modifica original)
for f in sorted(set(canasta)):
    print(f)
```

4.8 Operadores de comparacion y logicos

Operador	Descripcion
`in, not in`	Pertenencia en secuencias
`is, is not`	Identidad de objetos
`not, and, or`	Operadores booleanos (prioridad: not >and >or)
`:=`	Operador morsa (asignacion en expresion, Python 3.8+)

```
# Comparaciones encadenadas
a < b == c   # equivale a: (a < b) and (b == c)

# Evaluacion de cortocircuito
non_null = '' or 'Trondheim' or 'Hammer'  # 'Trondheim'
```

```python
# Operador morsa
if (n := len(lista)) > 10:
    print(f"Lista muy larga ({n} elementos)")
```

5. Modulos y Paquetes

5.1 Modulos

```
# Un modulo es un archivo .py con definiciones

# --- fibo.py ---
def fib(n):
    """Imprime serie Fibonacci hasta n."""
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a + b
    print()

def fib2(n):
    """Retorna lista Fibonacci hasta n."""
    resultado = []
    a, b = 0, 1
    while a < n:
        resultado.append(a)
        a, b = b, a + b
    return resultado
```

5.2 Formas de importar

```
# Importar modulo completo
import fibo
fibo.fib(1000)

# Importar nombres especificos
from fibo import fib, fib2
fib(500)

# Importar con alias
import fibo as f
f.fib(500)

from fibo import fib as fibonacci
fibonacci(500)

# Importar todo (NO recomendado en produccion)
from fibo import *
```

5.3 Ejecutar modulos como scripts

```
# Patron comun para modulos ejecutables
if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))

# Uso: python fibo.py 50
# Al importar, este codigo NO se ejecuta
```

5.4 Ruta de busqueda de modulos

```python
import sys
print(sys.path)
# 1. Directorio del script actual
# 2. PYTHONPATH (variable de entorno)
# 3. Directorio de instalacion por defecto

# Agregar rutas personalizadas
sys.path.append('/ruta/a/mis/modulos')
```

5.5 Paquetes

```python
# Estructura de paquetes:
# sonido/
#     __init__.py            (inicializa el paquete)
#     formatos/
#         __init__.py
#         wavread.py
#         mp3read.py
#     efectos/
#         __init__.py
#         eco.py
#         reverb.py

# Importar desde paquetes
import sonido.efectos.eco
sonido.efectos.eco.aplicar_eco(audio)

from sonido.efectos import eco
eco.aplicar_eco(audio)

from sonido.efectos.eco import aplicar_eco
aplicar_eco(audio)

# Importaciones relativas
from . import eco               # paquete actual
from .. import formatos         # paquete padre
from ..formatos import wavread  # desde paquete hermano
```

5.6 La funcion dir()

```python
import fibo
dir(fibo)     # ['__name__', 'fib', 'fib2']

# Sin argumentos: nombres definidos actualmente
dir()         # ['__builtins__', '__name__', 'fibo', ...]

# Ver funciones integradas
import builtins
dir(builtins)
```

6. Entrada y Salida

6.1 F-strings (cadenas formateadas)

```
nombre = "Ana"
edad = 25
import math

# Basico
f'Hola {nombre}, tienes {edad} anos'

# Con formato numerico
f'Pi es aproximadamente {math.pi:.3f}'       # 3.142
f'Pi es aproximadamente {math.pi:10.3f}'     # '     3.142'

# Alineacion y ancho
f'{"Nombre":10} {"Telefono":>10}'
f'{"Ana":10} {4098:10d}'

# Conversiones
f'{animal!r}'     # repr()
f'{animal!s}'     # str()
f'{animal!a}'     # ascii()

# Autodocumentacion (Python 3.8+)
f'{nombre=} {edad=}'   # "nombre='Ana' edad=25"
```

6.2 str.format()

```
# Posicional
'{0} y {1}'.format('spam', 'huevos')

# Por nombre
'Esto es {comida}'.format(comida='spam')

# Con diccionarios
tabla = {'Ana': 4127, 'Juan': 4098}
'Ana: {Ana:d}; Juan: {Juan:d}'.format(**tabla)

# Tabla formateada
for x in range(1, 11):
    print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
```

6.3 Formateo manual

```
str.rjust(ancho)      # alinear a la derecha
str.ljust(ancho)      # alinear a la izquierda
str.center(ancho)     # centrar
str.zfill(ancho)      # rellenar con ceros

'12'.zfill(5)         # '00012'
'-3.14'.zfill(7)      # '-003.14'
```

6.4 Lectura y escritura de archivos

```
# SIEMPRE usar 'with' (cierra automaticamente el archivo)
with open('archivo.txt', 'r', encoding='utf-8') as f:
```

```
    contenido = f.read()

# Modos de apertura
'r'   # lectura (defecto)
'w'   # escritura (sobreescribe)
'a'   # agregar al final
'r+'  # lectura y escritura
'rb'  # lectura binaria
'wb'  # escritura binaria
```

6.4.1 Metodos de archivos

```
# Leer todo el contenido
f.read()              # retorna string completo
f.read(10)            # retorna 10 caracteres

# Leer linea por linea
f.readline()          # una linea (incluye \n)

# Leer todas las lineas
f.readlines()         # retorna lista de lineas

# Iterar linea por linea (eficiente en memoria)
for linea in f:
    print(linea, end='')

# Escribir
f.write('Hola mundo\n')    # retorna num de caracteres escritos

# Posicion en archivo
f.tell()              # posicion actual
f.seek(0)             # ir al inicio
f.seek(5)             # ir al byte 5
```

6.5 JSON

```
import json

# Serializar a string JSON
datos = [1, 'simple', 'lista']
json_str = json.dumps(datos)   # '[1, "simple", "lista"]'

# Deserializar desde string JSON
obj = json.loads('{"clave": "valor"}')

# Guardar en archivo
with open('datos.json', 'w', encoding='utf-8') as f:
    json.dump(datos, f)

# Leer desde archivo
with open('datos.json', 'r', encoding='utf-8') as f:
    datos = json.load(f)
```

7. Errores y Excepciones

7.1 Tipos de errores

Tipo	Causa
`SyntaxError`	Error de sintaxis (detectado al parsear)
`NameError`	Variable no definida
`TypeError`	Operacion con tipo incorrecto
`ValueError`	Valor inapropiado para el tipo
`ZeroDivisionError`	Division por cero
`IndexError`	Indice fuera de rango
`KeyError`	Clave no encontrada en diccionario
`FileNotFoundError`	Archivo no encontrado
`ImportError`	Error al importar modulo
`AttributeError`	Atributo no encontrado
`IOError / OSError`	Error de entrada/salida

7.2 try / except / else / finally

```
try:
    # Codigo que puede generar excepcion
    x = int(input("Numero: "))
    resultado = 10 / x
except ValueError:
    # Se ejecuta si hay ValueError
    print("No es un numero valido")
except ZeroDivisionError:
    # Se ejecuta si hay ZeroDivisionError
    print("No se puede dividir por cero")
except (RuntimeError, TypeError) as e:
    # Multiples excepciones + acceso al error
    print(f"Error: {e}")
else:
    # Se ejecuta si NO hubo excepcion
    print(f"Resultado: {resultado}")
finally:
    # Se ejecuta SIEMPRE (ideal para limpieza)
    print("Operacion terminada")
```

7.3 Lanzar excepciones

```
# raise lanza una excepcion
raise ValueError("Valor invalido")

# Re-lanzar la excepcion actual
try:
    algo()
except Exception:
    print("Ocurrio un error")
    raise   # re-lanza la misma excepcion
```

7.4 Encadenamiento de excepciones

```
# Encadenamiento explicito con 'from'
try:
```

```
    conectar()
except ConnectionError as exc:
    raise RuntimeError('Fallo la conexion') from exc

# Desactivar encadenamiento automatico
raise RuntimeError('Error') from None
```

7.5 Excepciones personalizadas

```
class MiError(Exception):
    """Excepcion personalizada para mi aplicacion."""
    pass

class ErrorDeValidacion(Exception):
    def __init__(self, mensaje, campo):
        self.mensaje = mensaje
        self.campo = campo
        super().__init__(self.mensaje)

# Uso
raise ErrorDeValidacion("Campo requerido", "nombre")
```

7.6 Sentencia with (limpieza predefinida)

```
# Garantiza limpieza automatica de recursos
with open("archivo.txt") as f:
    for linea in f:
        print(linea, end="")
# El archivo se cierra automaticamente, incluso si hay error
```

7.7 Grupos de excepciones (Python 3.11+)

```
# ExceptionGroup agrupa multiples excepciones
def f():
    raise ExceptionGroup('errores', [
        OSError('error 1'),
        SystemError('error 2')
    ])

# except* maneja tipos especificos dentro del grupo
try:
    f()
except* OSError as e:
    print("Errores de OS")
except* SystemError as e:
    print("Errores de sistema")

# Agregar notas a excepciones
try:
    raise TypeError('tipo malo')
except Exception as e:
    e.add_note('Informacion adicional')
    raise
```

8. Clases y POO

8.1 Ambitos y espacios de nombres (LEGB)

```
# Python busca nombres en este orden:
# L - Local (funcion actual)
# E - Enclosing (funciones externas)
# G - Global (modulo)
# B - Built-in (funciones integradas)

x = "global"

def externa():
    x = "enclosing"
    def interna():
        x = "local"
        print(x)          # "local"
    interna()

# nonlocal modifica variable de funcion externa
def externa():
    x = "enclosing"
    def interna():
        nonlocal x
        x = "modificado"
    interna()
    print(x)   # "modificado"

# global modifica variable del modulo
def funcion():
    global x
    x = "global modificado"
```

8.2 Definicion de clases

```
class MiClase:
    """Documentacion de la clase."""

    # Variable de clase (compartida por todas las instancias)
    contador = 0

    def __init__(self, nombre, edad):
        # Variables de instancia (unicas por instancia)
        self.nombre = nombre
        self.edad = edad
        MiClase.contador += 1

    def saludar(self):
        return f"Hola, soy {self.nombre}"

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

# Crear instancias
a = MiClase("Ana", 25)
b = MiClase("Bob", 30)
print(a.saludar())      # "Hola, soy Ana"
print(MiClase.contador)  # 2
```

Error comun: Variables de clase mutables se comparten entre instancias:

```
# MAL - lista compartida
class Perro:
    trucos = []  # compartida!
    def agregar(self, truco):
        self.trucos.append(truco)

# BIEN - lista por instancia
class Perro:
    def __init__(self, nombre):
        self.trucos = []  # unica por instancia
    def agregar(self, truco):
        self.trucos.append(truco)
```

8.3 Herencia

```
class Animal:
    def __init__(self, nombre):
        self.nombre = nombre

    def hablar(self):
        raise NotImplementedError

class Perro(Animal):
    def hablar(self):
        return f"{self.nombre} dice Guau!"

class Gato(Animal):
    def hablar(self):
        return f"{self.nombre} dice Miau!"

# Polimorfismo
animales = [Perro("Rex"), Gato("Misi")]
for a in animales:
    print(a.hablar())

# Verificar tipos
isinstance(rex, Perro)      # True
isinstance(rex, Animal)     # True
issubclass(Perro, Animal)   # True
```

8.4 Herencia multiple

```
class Base1:
    def metodo(self):
        print("Base1")

class Base2:
    def metodo(self):
        print("Base2")

class Derivada(Base1, Base2):
    pass

d = Derivada()
d.metodo()      # "Base1" (busqueda izquierda-derecha)
```

```
# super() para herencia cooperativa
class D(Base1, Base2):
    def metodo(self):
        super().metodo()  # sigue el MRO
```

8.5 Variables privadas

```
# Convencion: _ indica "uso interno"
class MiClase:
    def __init__(self):
        self._protegido = 1     # convencion, accesible
        self.__privado = 2      # name mangling -> _MiClase__privado

# __nombre se transforma a _Clase__nombre
# Previene colisiones accidentales en subclases
```

8.6 Iteradores

```
# Protocolo: __iter__() retorna objeto con __next__()

class Reversa:
    def __init__(self, datos):
        self.datos = datos
        self.indice = len(datos)

    def __iter__(self):
        return self

    def __next__(self):
        if self.indice == 0:
            raise StopIteration
        self.indice -= 1
        return self.datos[self.indice]

for c in Reversa('spam'):
    print(c)   # m, a, p, s
```

8.7 Generadores

```
# Funciones que usan yield (crean iteradores automaticamente)
def reversa(datos):
    for i in range(len(datos)-1, -1, -1):
        yield datos[i]

for c in reversa('golf'):
    print(c)   # f, l, o, g

# Expresiones generadoras (como list comprehension con parentesis)
sum(i*i for i in range(10))         # 285
max(student.gpa for student in graduates)
```

8.8 Dataclasses

```
from dataclasses import dataclass
```

```python
@dataclass
class Empleado:
    nombre: str
    departamento: str
    salario: int

juan = Empleado('Juan', 'TI', 50000)
print(juan.nombre)       # 'Juan'
print(juan.salario)      # 50000
```

9. Biblioteca Estandar – Parte I

9.1 os – Interfaz del sistema operativo

```
import os
os.getcwd()              # directorio actual
os.chdir('/ruta')        # cambiar directorio
os.system('mkdir nueva') # ejecutar comando del sistema

import shutil
shutil.copyfile('origen.txt', 'destino.txt')
shutil.move('/origen', '/destino')
```

9.2 glob – Comodines de archivos

```
import glob
glob.glob('*.py')        # ['main.py', 'utils.py', ...]
glob.glob('**/*.py', recursive=True)  # busqueda recursiva
```

9.3 sys – Parametros del sistema

```
import sys
sys.argv          # argumentos de linea de comandos
sys.stdin         # entrada estandar
sys.stdout        # salida estandar
sys.stderr        # salida de errores
sys.path          # ruta de busqueda de modulos
sys.exit()        # terminar el programa
```

9.4 argparse – Procesamiento de argumentos

```
import argparse
parser = argparse.ArgumentParser(description='Mi programa')
parser.add_argument('archivo', help='Archivo de entrada')
parser.add_argument('-n', '--lineas', type=int, default=10)
args = parser.parse_args()
print(args.archivo, args.lineas)
```

9.5 re – Expresiones regulares

```
import re
re.findall(r'\bf[a-z]*', 'foot fell fastest')
# ['foot', 'fell', 'fastest']

re.sub(r'(\b[a-z]+) \1', r'\1', 'cat in the the hat')
# 'cat in the hat'

# Para operaciones simples, preferir metodos de string
'tea for too'.replace('too', 'two')
```

9.6 math – Funciones matematicas

```
import math
math.pi              # 3.141592653589793
math.e               # 2.718281828459045
```

```
math.cos(math.pi/4)  # 0.7071067811865476
math.log(1024, 2)    # 10.0
math.sqrt(16)        # 4.0
math.factorial(5)    # 120
math.gcd(12, 8)      # 4
```

9.7 random – Numeros aleatorios

```
import random
random.choice(['a', 'b', 'c'])       # elemento aleatorio
random.sample(range(100), 10)         # 10 elementos sin repetir
random.random()                       # float en [0.0, 1.0)
random.randrange(6)                   # entero en range(6)
random.shuffle(lista)                 # mezclar lista in-place
random.randint(1, 100)               # entero entre 1 y 100
```

9.8 statistics – Estadisticas basicas

```
import statistics
datos = [2.75, 1.75, 1.25, 0.25, 0.5, 1.25, 3.5]
statistics.mean(datos)      # media: 1.607...
statistics.median(datos)    # mediana: 1.25
statistics.variance(datos)  # varianza: 1.372...
statistics.stdev(datos)     # desviacion estandar
```

9.9 datetime – Fechas y tiempos

```
from datetime import date, datetime, timedelta

hoy = date.today()
hoy.strftime("%d/%m/%Y")    # '02/02/2026'

nacimiento = date(1990, 5, 15)
edad = hoy - nacimiento
print(edad.days)              # dias vividos

# Operaciones con tiempo
manana = hoy + timedelta(days=1)
```

9.10 zlib – Compresion de datos

```
import zlib
s = b'datos repetidos repetidos repetidos'
comprimido = zlib.compress(s)
len(s)           # 37
len(comprimido)  # mas pequeno
zlib.decompress(comprimido)  # datos originales
```

9.11 timeit – Medicion de rendimiento

```
from timeit import Timer
Timer('a,b = b,a', 'a=1; b=2').timeit()
# Mide tiempo de ejecucion
```

9.12 doctest y unittest – Control de calidad

```
# doctest - pruebas en docstrings
def promedio(valores):
    """Calcula el promedio.

    >>> promedio([20, 30, 70])
    40.0
    """
    return sum(valores) / len(valores)

import doctest
doctest.testmod()

# unittest - suite de pruebas completa
import unittest

class TestPromedio(unittest.TestCase):
    def test_basico(self):
        self.assertEqual(promedio([20, 30, 70]), 40.0)

    def test_division_cero(self):
        with self.assertRaises(ZeroDivisionError):
            promedio([])

unittest.main()
```

10. Biblioteca Estandar – Parte II

10.1 Formato de salida avanzado

```
# reprlib - representaciones abreviadas
import reprlib
reprlib.repr(set('supercalifragilistic'))
# "{'a', 'c', 'f', 'g', 'i', ...}"

# pprint - impresion elegante
import pprint
pprint.pprint(datos_complejos, width=40)

# textwrap - ajuste de texto
import textwrap
print(textwrap.fill(texto_largo, width=60))

# locale - formato regional de numeros
import locale
locale.format_string("%d", 1234567, grouping=True)
# '1,234,567'
```

10.2 Plantillas de texto

```
from string import Template
t = Template('$quien le envio $$10 a $destino')
t.substitute(quien='Ana', destino='Juan')
# 'Ana le envio $10 a Juan'

# safe_substitute no lanza error si falta un valor
t.safe_substitute(quien='Ana')
# 'Ana le envio $10 a $destino'
```

10.3 struct – Datos binarios

```
import struct
# pack/unpack para formatos binarios
# '<' = little-endian, 'H' = unsigned short, 'I' = unsigned int
campos = struct.unpack('<IIIHH', datos_binarios)
```

10.4 threading – Multi-hilos

```
import threading

class MiHilo(threading.Thread):
    def __init__(self, archivo):
        threading.Thread.__init__(self)
        self.archivo = archivo

    def run(self):
        # Tarea en segundo plano
        procesar(self.archivo)

hilo = MiHilo('datos.txt')
hilo.start()          # iniciar hilo
hilo.join()           # esperar a que termine
```

Recomendacion: Usar el modulo `queue` para comunicacion entre hilos. Concentrar acceso a recursos en un solo hilo.

10.5 logging – Registro de eventos

```
import logging

# Niveles: DEBUG, INFO, WARNING, ERROR, CRITICAL
logging.debug('Informacion de depuracion')
logging.info('Mensaje informativo')
logging.warning('Advertencia: %s no encontrado', 'config')
logging.error('Error ocurrido')
logging.critical('Error critico')
# Por defecto solo se muestran WARNING y superiores
```

10.6 Herramientas para listas

```
# array - listas homogeneas compactas
from array import array
a = array('H', [4000, 10, 700])   # unsigned short

# collections.deque - cola de doble extremo
from collections import deque
d = deque(["t1", "t2", "t3"])
d.append("t4")
d.popleft()               # 't1'

# bisect - busqueda binaria en listas ordenadas
import bisect
bisect.insort(lista_ordenada, nuevo_elemento)

# heapq - heap (cola de prioridad)
from heapq import heapify, heappush, heappop
datos = [5, 3, 8, 1]
heapify(datos)            # [1, 3, 8, 5]
heappush(datos, 2)
heappop(datos)            # 1 (siempre el menor)
```

10.7 decimal – Aritmetica decimal precisa

```
from decimal import Decimal, getcontext

# Precision exacta para finanzas
round(Decimal('0.70') * Decimal('1.05'), 2)  # Decimal('0.74')
round(0.70 * 1.05, 2)                         # 0.73 (impreciso!)

# Comparaciones exactas
sum([Decimal('0.1')] * 10) == Decimal('1.0')  # True
0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 == 1.0  # False

# Precision arbitraria
getcontext().prec = 36
Decimal(1) / Decimal(7)
# Decimal('0.142857142857142857142857142857142857')
```

11. Entornos Virtuales y Gestion de Paquetes

11.1 Por que usar entornos virtuales

Los entornos virtuales resuelven conflictos de dependencias permitiendo que cada proyecto tenga sus propias versiones de paquetes aisladas del sistema global.

11.2 Crear y activar entornos virtuales

```
# Crear entorno virtual
python -m venv mi_entorno
python -m venv .venv          # convencion: .venv

# Activar
# Linux/macOS:
source mi_entorno/bin/activate

# Windows:
mi_entorno\Scripts\activate

# El prompt cambia para indicar el entorno activo:
# (mi_entorno) $

# Desactivar
deactivate
```

11.3 Gestion de paquetes con pip

```
# Instalar paquete (ultima version)
pip install requests

# Instalar version especifica
pip install requests==2.28.0

# Actualizar paquete
pip install --upgrade requests

# Desinstalar
pip uninstall requests

# Ver informacion de un paquete
pip show requests

# Listar paquetes instalados
pip list

# Exportar dependencias a archivo
pip freeze > requirements.txt

# Instalar desde archivo de dependencias
pip install -r requirements.txt
```

Buena practica: Incluir `requirements.txt` en el control de versiones para reproducir el entorno facilmente.

12. Guia de Estilo PEP 8

Regla	Ejemplo
Indentacion: 4 espacios	`if True: hacer_algo()`
Longitud maxima de linea: 79 caracteres	Usar parentesis para continuar lineas
Lineas en blanco: separar funciones y clases	2 lineas entre funciones de nivel superior
Imports al inicio del archivo	Un import por linea
Espacios alrededor de operadores	`a = f(1, 2) + g(3)`
Clases: NotacionCamelCase	`class MiClase:`
Funciones/variables: snake_case	`def mi_funcion():`
Constantes: MAYUSCULAS	`MAX_INTENTOS = 5`
Primer argumento de metodo: self	`def metodo(self):`
Codificacion: UTF-8	Predeterminado en Python 3
Docstrings: usar siempre	`"Descripcion breve."`
Comparar con None usando is	`if x is None:`

12.0.1 Orden de imports

```
# 1. Modulos de la biblioteca estandar
import os
import sys

# 2. Modulos de terceros
import requests
import numpy

# 3. Modulos locales/del proyecto
from mi_paquete import mi_modulo
```

Referencia Rapida – Cheat Sheet

Tipos de datos fundamentales

Tipo	Ejemplo	Mutable	Ordenado
`int`	`42`	No	N/A
`float`	`3.14`	No	N/A
`str`	`"hola"`	No	Si
`bool`	`True, False`	No	N/A
`list`	`[1, 2, 3]`	Si	Si
`tuple`	`(1, 2, 3)`	No	Si
`set`	`{1, 2, 3}`	Si	No
`dict`	`{."a": 1}`	Si	Insercion
`NoneType`	`None`	No	N/A

Operadores

Aritmeticos	
`+, -, *, /`	Basicos
`//`	Division entera
`%`	Modulo
`*`	Potencia

Comparacion	
`==, !=`	Igualdad
`<, >, <=, >=`	Orden
`is, is not`	Identidad
`in, not in`	Pertenencia

Funciones integradas mas usadas

Funcion	Descripcion
`print()`	Imprimir en consola
`input()`	Leer entrada del usuario
`len()`	Longitud de secuencia
`type()`	Tipo del objeto
`int(), float(), str()`	Conversion de tipos
`list(), tuple(), set(), dict()`	Crear colecciones
`range()`	Generar secuencia numerica
`enumerate()`	Indice + valor en iteracion
`zip()`	Emparejar secuencias
`map(), filter()`	Aplicar/filtrar con funcion
`sorted()`	Retornar lista ordenada
`reversed()`	Iterador invertido
`sum(), min(), max()`	Operaciones agregadas
`abs(), round()`	Valor absoluto, redondeo
`isinstance()`	Verificar tipo de instancia
`hasattr(), getattr()`	Verificar/obtener atributos
`open()`	Abrir archivos
`help()`	Ayuda interactiva
`dir()`	Listar atributos de objeto

Patrones comunes

```
# Intercambio de variables
a, b = b, a
```

```python
# Operador ternario
x = "par" if n % 2 == 0 else "impar"

# Comprension de listas con condicion
pares = [x for x in range(20) if x % 2 == 0]

# Comprension de diccionarios
cuadrados = {x: x**2 for x in range(6)}

# Abrir archivo con with
with open('archivo.txt', 'r') as f:
    contenido = f.read()

# Main guard
if __name__ == "__main__":
    main()
```

Fuente: Tutorial Oficial de Python – https://docs.python.org/es/3/tutorial/

Generado para Academia Bio-Python – Febrero 2026