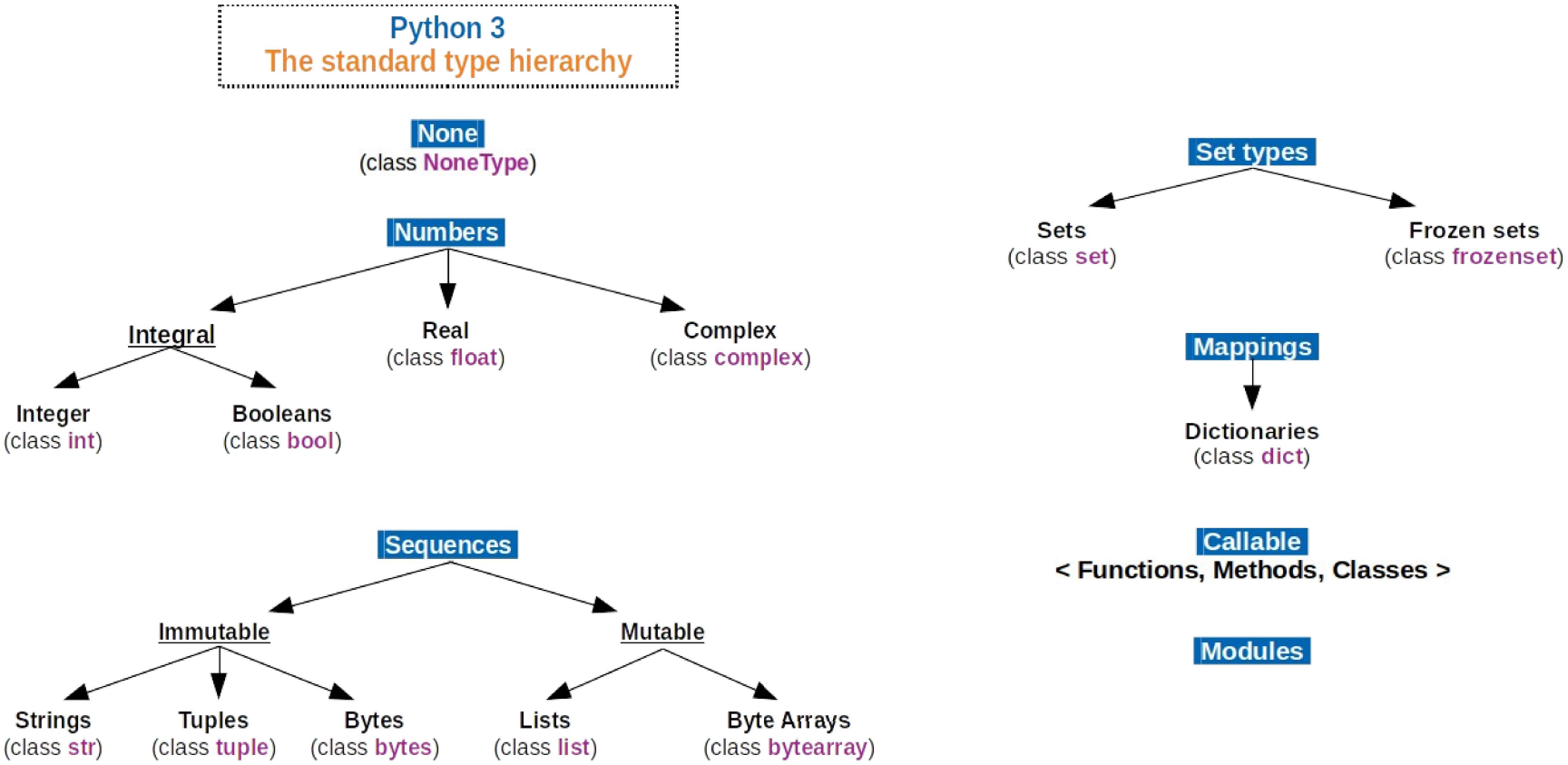
Tipo de datos

Tipo	Clase	Notas	Ejemplo
NoneType	None	Representa la ausencia de valor	None
bool	Numbers	Valor booleano verdadero o falso	True o False
int	Numbers	Número entero de tamaño ilimitado	42
float	Numbers	Número real; coma flotante	3.1415927
complex	Numbers	Número complejo con parte real y parte imaginaria j	4.5 + 3j
str	Sequences	Cadena en formato unicode. Inmutable	“Texto”
list	Sequences	Secuencia de datos, pueden ser de diversos tipos. Mutable.	[4.0, 'Cadena', True]
tuple	Sequences	Secuencia de datos, pueden ser de diversos tipos. Inmutable.	(4.0, 'Cadena', True)
set	Set Types	Conjunto de datos, sin orden, no contiene duplicados. Mutable	set([4.0, 'Cadena', True])
frozenset	Set Types	Conjunto de datos, sin orden, no contiene duplicados. Inmutable	frozenset([4.0, 'Cadena', True])
dict	Mappings	Diccionario de pares clave:valor (array asociativo)	{'key1': 1.0, 'key2': False}
bytearray	Binary Sequences	Secuencia de bytes. Mutable	bytearray([119, 105, 107, 105])
bytes	Binary Sequences	Secuencia de bytes. Inmutable	bytes([119, 105, 107, 105])

Jerarquía de Tipos de Datos



Tipos Numéricos

- Números enteros (int):
  - Decimal: 24, 60
  - Binario: 0b010011, 0b1101
  - Hexadecimal: 0x18, 0x3cf4
  - Octal: 0o30, 0o74
- Números de punto flotante (float): números reales y la precisión depende del equipo.
  - 3.141595
  - 12.
  - -45.3556
  - 2,0/3,0
- Números complejos (complex):
  - 6.32 + 45j
  - 0.117j
  - (2 + 0j)
  - 1j
- Valores booleanos (bool): Se usa para expresiones lógicas
  - False (equivale al número 0)
  - True (cualquier otro valor diferente de cero y 1 por defecto)

String y None

- **str** (cadena de caracteres):
  - 'Wikipedia'
  - "Wikipedia"
  - """Con
    múltiples
    líneas"""
- **None**:
  - El tipo None representa un valor "vació".
  - a = None

Listas

- **Listas (array indexado):**
  - Es la secuencia más general en python.
  - Mutables; se puede cambiar su contenido en tiempo de ejecución.
  - Para declarar una lista se usan los corchetes [] y los elementos se separan por comas.
  - Pueden contener elementos de diferentes tipos.
  - No tienen un tamaño fijo.
  - Los elementos son ordenados por la posición.
  - Para acceder a los elementos se utiliza un índice entero (empezando por "0", no por "1"). Se pueden utilizar índices negativos para acceder elementos a partir del final.

Listas

- Crear una lista:

```
lista = ["abc", 42, 3.1415]
```

- Acceder a un elemento por su índice:

```
lista[0]
'abc'
```

- Acceder a un elemento usando un índice negativo:

```
lista[-1]
3.1415
```

- Añadir un elemento al final de la lista:

```
lista.append(True)
lista
['abc', 42, 3.1415, True]
```

Listas

- Re-asignar el valor del primer elemento de la lista:

```
lista[0] = "xyz"
```

- Borrar un elemento de la lista:

```
lista.remove(True)
del lista[0]
```

- Mostrar una sublista:

```
lista[0:2]     # Del índice "0" al "2" (sin incluir este último)
['xyz', 42]
```

- Listas anidadas (una dentro de otra):

```
lista_anidada = [lista, [True, 42]]
lista_anidada
[['xyz', 42, 3.1415], [True, 42]]
lista_anidada[1][0]
True
```

Listas

```
lista = [22, True, "a list", [1, 2]]
lista[0] ⇒ ?
lista[2][4] ⇒ ?
lista[-1][-2] ⇒ ?
lista[0:3] ⇒ ?
lista[:3] ⇒ ?
lista.append('DevOps') ⇒ ?
lista.insert (0,“a list”) ⇒ ?
lista.remove(“a list”) ⇒ ?
```

Listas

```
lista = [22, True, "a list", [1, 2]]
lista[0] ⇒ 22
lista[2][4] ⇒ 's'
lista[-1][-2] ⇒ 1
lista[0:3] ⇒ [22, True, 'a list']
lista[:3] ⇒ [22, True, 'a list']
lista.append('DevOps') ⇒ [22, True, 'a list', [1, 2], 'DevOps']
lista.insert (0,“a list”) ⇒ ['a list', 22, True, 'a list', [1, 2], 'DevOps']
lista.remove(“a list”) ⇒ [22, True, 'a list', [1, 2], 'DevOps']
```

Tuplas

- **Tuplas:**
  - Es otra secuencia en python como las listas.
  - **Inmutables**; no se puede cambiar su contenido en tiempo de ejecución.
  - Para declarar una lista se usan los **paréntesis ()** y los elementos se separan por comas. **Es necesario que tengan como mínimo una coma. También se pueden declarar sin los paréntesis.**
  - Pueden contener elementos de diferentes tipos.
  - Pueden definirse de cualquier tamaño.
  - Los elementos son ordenados por la posición.
  - Para acceder a los elementos se utiliza un índice entero (empezando por "0", no por "1"). Se pueden utilizar índices negativos para acceder elementos a partir del final.

Tuplas

- Crear una tupla:

  tupla = ("abc", 42, 3.1415)
- Acceder a un elemento por su índice:
  - tupla[0]

    'abc'
- Acceder a un elemento usando un índice negativo:
  - tupla[-1]

    3.1415
- No es posible modificar la tupla:
  - del tupla[0]

    (Excepción)
  - tupla[0] = "xyz"

    (Excepción)

Tuplas

- Mostrar una sub-tupla:

```
tupla[0:2]   # Del índice "0" al "2" (sin incluir este último)
('abc', 42)
```

- Tuplas anidadas (una dentro de otra):

```
tupla_anidada = (tupla, (True, 3.1415))
(('abc', 42, 3.1415), (True, 3.1415))
tupla_anidada[1][0]
True
```

- También es una tupla:

```
1, 2, 3, "abc"
(1,)  #Ojo (1) no es una tupla
(1, 2,)
```

Tuplas

- La inmutabilidad se puede omitir si una nueva estructura es enlazada a la tupla original
  - >>> t = 10,15,20
  - >>> t = t[0],t[2]
  - >>> t
  - (10,20)

Diccionarios

- **Diccionarios (array asociativo):**
  - **Mutables**; se puede cambiar su contenido en tiempo de ejecución.
  - Para declarar un diccionario se usan las llaves **{}**. Contienen elementos separados por comas, donde cada elemento está formado por un par **clave:valor** (el símbolo : separa la clave de su valor correspondiente).
  - Las claves de un diccionario deben ser inmutables (strings, números, o tuplas)
  - El valor asociado a una clave puede ser de cualquier tipo de dato, incluso un diccionario.
  - No tienen un tamaño fijo.
  - Indexados por la clave.

Diccionarios

- Crear un diccionario:

  diccionario = {"cadena": "abc", "numero": 42, "lista": [True, 42]}

- Acceder a un elemento por su clave:
  - diccionario["cadena"]

    'abc'
  - diccionario["lista"][0]

    True
- Insertar un nuevo elemento clave:valor:

  diccionario["decimal"] = 3.1415927

Diccionarios

- Re-asignar el valor del primer elemento de la lista:

```
diccionario["cadena"] = "xyz"
```

- Borrar un elemento de la lista:
  - del diccionario["cadena"]
- También es posible que un valor sea un diccionario

```
diccionario_mixto = {"tupla": (True, 3.1415), "diccionario": diccionario}
diccionario_mixto["diccionario"]["lista"][1]
42
```

Diccionarios

```
dic = {'e': 2.718, 'pi': 3.141, 'fi': 1.618}
>>> dic['e']
?
# Actualizar el valor de pi a 3.141592
>>> ?
>>> dic.keys()
?
>>> dic.values()
?
>>> dic.items()
?
```

Diccionarios

```
dic = {'e': 2.718, 'pi': 3.141, 'fi': 1.618}
>>> dic['e']
2.718
# Actualizar el valor de pi a 3.141592
>>> dic['pi'] = 3.141592
>>> dic.keys()
dict_keys(['e', 'pi', 'fi'])
>>> dic.values()
dict_values([2.718, 3.141, 1.618])
>>> dic.items()
dict_items([('e', 2.718), ('pi', 3.141), ('fi', 1.618)])
```

Diccionarios

- Operaciones comunes con Diccionarios:
  - Agregar un elemento (update):
    - dic.update({"d":4})
  - Crear una copia del diccionario (copy):
    - nuevodic = dic.copy()
  - Eliminar todos los elementos de un diccionario:
    - dic.clear()

Conjuntos

- **Conjuntos:**
  - Los conjuntos se construyen mediante **set(items) / frozenset(items)** donde items es cualquier objeto iterable, como listas o tuplas.
    - set para conjuntos mutables y
    - frozenset para conjuntos inmutables.
  - Los conjuntos no mantienen el orden ni contienen elementos duplicados.
  - Se suelen utilizar para eliminar duplicados de una secuencia, o para operaciones matemáticas como intersección, unión, diferencia y diferencia simétrica.

Conjuntos

- Crear conjuntos:

  conjunto1 = set(["a", "b", "a"])

  conjunto2 = set(["a", "b", "c", "d"])

  conjunto_inmutable = frozenset(["a", "b", "a"])

- Intersección

  conjunto1 & conjunto2

  set(['a', 'b'])

- Unión

  conjunto1 | conjunto2

  set(['a', 'c', 'b', 'd'])

Conjuntos

- Diferencia (1)

  conjunto1 - conjunto2

  set([])

- Diferencia (2)

  conjunto2 - conjunto1

  set(['c', 'd'])

- Diferencia simétrica

  conjunto1 ^ conjunto2

  set(['c', 'd'])

Bytes, Bytearray

- Para manejar datos binarios python incluye los tipos bytes y bytearray.
- El tipo bytes es una secuencia inmutable de bytes, conceptualmente similar a una cadena. Y el tipo bytearray es una secuencia mutable de bytes.
- Representan a un carácter conforme a su número correspondiente en el código ASCII y se definen anteponiendo la letra b a los apostrofes o comillas.

  b'<texto>'

  b"<texto>"

Bytes

- Ejemplos:

```
palabra = b"Hola" // palabra =bytes([72,111,108,97])
palabra[0]
72
palabra[2:4]
b'la'
```

- str -> bytes.

```
bytes('hola', "utf-8")
```

- bytes -> str

```
str(b'hola'[1:3], 'ascii')
```

Bytearray

- Crear un bytearray desde un bytes:

```
x = bytearray(b"Python Bytes")
```

- Crear un bytearray desde un string:
  - `x = bytearray("Python Bytes", "utf8")`
- Crear un bytearray desde una lista de enteros:
  - `x = bytearray([94, 91, 101, 125, 111, 35, 120])`

Rebanadas de secuencias (slice)

- secuencia[x:y:z]
  - Desde x, hasta y sin incluir dicha posición y con incrementos de z
- Ejemplos de t=(1,2,3,4,5)
  - t[2:4] → Desde 2 hasta 3
  - t[:3] → Desde el comienzo hasta 2
  - t[3:] → Desde 3 hasta el final
  - t[:] → Desde el comienzo hasta el final
  - t[::-1] → En sentido inverso
  - t[::-2] → En sentido inverso con incrementos de 2