Licenciatura en Ciencia de Datos

Algoritmos II

Type Hints en Python

-**Extensión** al lenguaje que nos permitirá trabajar con Python como si fuese un **lenguaje tipado**

- Característica introducida en **Python 3.5**

- **Anotaciones opcionales** que se pueden añadir a las **variables**, **parámetros de funciones y valores de retorno** para indicar el **tipo de datos que se espera**

-**¡¡¡Python SIGUE SIENDO DE TIPADO DINÁMICO!!!**

-Los type hints son útiles para **documentar el código** y **mejorar la legibilidad**

-**No se aplican en tiempo de ejecución:** son **puramente informativos** y de ayuda durante el desarrollo.

Sintaxis básica de Type Hints

-**Anotaciones de tipo** utilizando dos puntos **(:) después del nombre de la variable o parámetro**, seguido del **tipo de dato** esperado. Para un **tipo de dato de retorno** se utiliza **-> al final de la firma**

```
def potencia(base: float, exponente: int) -> float:
    return base ** exponente


potencia(10, 2)      # 100
```

Comprobando el código

-Herramientas externas como **mypy:**

pip install mypy

mypy mi_aplicacion.py

Colecciones

Listas:

```
xs: list[int] = [1, 3, 4]
ys: list[str] = ['a', 'b', 'c']
```

Sets:

```
s1: set[int] = {1, 2, 4}
s2: set[str] = {'a', 'b', 'c'}
s3: set[list[int]] = {[1,2], [3,5]}     # TypeError: unhashable type: 'list';  set requiere que sus elementos sean hasheables, las listas no lo son
s3: set[tuple[int]] = {(1,2), (3,5)}    # ok
```

Colecciones(cont)

Diccionarios:

```
d1: dict[str, float] = {'a': 2.1, 'b': 3.4}
d2: dict[int, list[str]] = {1: ['a','b'], 2: ['c','d']}
d3: dict[list[int], int] = {[1,2]: 0, [3,5]: 1}     # TypeError: unhashable type:
'list'
d3: dict[tuple[int, ...], int] = {(1,2): 0, (3,5): 1}   # ok
```

Tipos de construcción

-Python ofrece un **conjunto de tipos predefinidos**

Any:

-indica que una variable, parámetro de función o valor de retorno puede ser de **cualquier tipo**.

```
from typing import Any

def imprimir_cualquier_cosa(valor: Any):  #acepta un parámetro valor de cualquier tipo y lo imprime
    print(valor)

# Ejemplos de uso
imprimir_cualquier_cosa(123)        # Imprime: 123
imprimir_cualquier_cosa("Hola")     # Imprime: Hola
imprimir_cualquier_cosa([1, 2, 3]) # Imprime: [1, 2, 3]
```

Tipos de construcción

Union:

- identificar **tipos de datos** que son **subtipos** de al menos uno de los tipos incluidos en la **unión**

```
from typing import Union
def mi_funcion(x: Union[float, str]):
    pass

mi_funcion(4)            # ok
mi_funcion(3.6)          # ok
mi_funcion('prueba')     # ok
```

También se puede con **operador |:**

```
def potencia(base: int | float, exponente: int | float) -> int | float:
    return base ** exponente
```

Tipos de construcción

Optional:

-**Optional[X]** es análogo a **X | None** o **Union[X, None]**. Se representa la **posibilidad de tener un valor o no**

```
from typing import Optional

def division(x: int, y: int) -> Optional[float]:
    if y == 0:
        return None
    return x / y

division(9, 4)      # 2.25
division(10, 0)     # None
```

Tipos de construcción

Callable:

-Indica **funciones** y el resto de **objetos invocables** (implementan el método especial **__call__()**)

Callable[[tipo_param_1, tipo_param_2, tipo_param_3], tipo_retorno]

-Los **tipos de datos de los parámetros** se declaran como una **lista de tipos** que se sitúa como **primer elemento**, el **segundo** es el **tipo de dato de retorno**

Tipos de construcción

```python
from typing import Callable

# Definición de una función que acepta otra función como argumento
def aplicar_operacion(x: int, y: int, operacion: Callable[[int, int], int]) -> int:
    return operacion(x, y)

# Definición de algunas funciones que coinciden con la firma esperada
def suma(a: int, b: int) -> int:
    return a + b

def producto(a: int, b: int) -> int:
    return a * b

# Uso de la función aplicar_operacion con diferentes operaciones
resultado_suma = aplicar_operacion(3, 4, suma)
print(resultado_suma)  # Salida: 7

resultado_producto = aplicar_operacion(3, 4, producto)
print(resultado_producto)  # Salida: 12
```

Tipos de construcción

Tuple:

-Se indica **un tipo de dato por elemento** de la tupla. Si el resto son del mismo tipo que el primero se usa **...**

```
t1: tuple[int, int] = (1, 2)
t2: tuple[str, int] = ('a', 3)
t3: tuple = ('a', 2, 3, 1)
t4: tuple[int, ...] = (1, 2, 3, 4, 5, 6)
t4 = (1, 2)
```

Generics

- **NO hay un SOPORTE NATIVO** para la **definición de TIPOS DE DATOS GENÉRICOS** como parte del lenguaje.

-Existen **TÉCNICAS** y **CONVENCIONES:** una forma es a través de los **type hints**

```python
from typing import TypeVar, Generic

T = TypeVar('T') # Definimos un tipo genérico T

class Caja(Generic[T]): # Creamos una clase genérica que puede manejar cualquier tipo T
    def __init__(self, contenido: T):
        self.contenido = contenido
    def obtener_contenido(self) -> T:
        return self.contenido
    def __str__(self) -> str:
        return f"Caja contiene: {self.contenido}"

# Uso de la clase genérica Caja con diferentes tipos
caja_cadenas = Caja[str]("Hola Mundo") # Caja de cadenas
print(caja_cadenas)  # Caja contiene: Hola Mundo
caja_flotantes = Caja[float](3.14)
print(caja_flotantes)  # Caja contiene: 3.14 # Caja de flotantes
```