

Seminario de Lenguajes - Python

Control de tipos

Control de tipos en Python

Antes de empezar: ¿para qué nos sirve conocer el tipo de datos de una variable?

- Los tipos de datos nos permiten relacionar un **conjunto de valores** que son de ese tipo con las **operaciones** que se pueden aplicar sobre esos valores.

¿Qué es un sistema de tipos?

- El sistema de tipos es un conjunto de reglas que tiene un lenguaje que nos permite manipular los datos de nuestros programas.
- Incluyen las conversiones explícitas e implícitas que podemos realizar.

Lenguajes con tipado estático vs. dinámico

- Se refiere a si el tipo de una variable se conoce en tiempo de compilación o en ejecución.

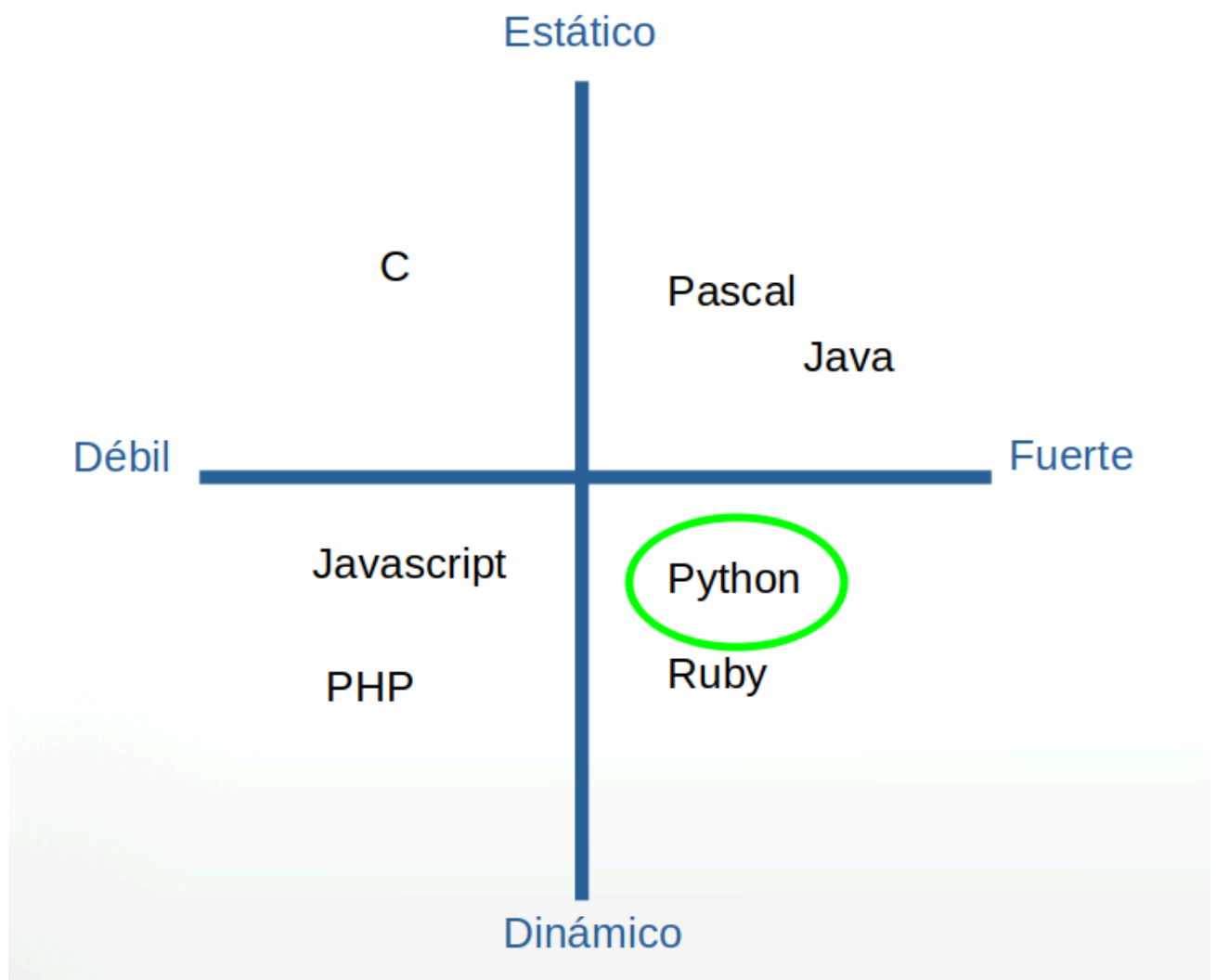
```
In [ ]: x = "casa"  
        type(x)
```

Lenguajes fuertemente tipados vs. débilmente tipados

- **Fuertemente tipados:** no se puede aplicar operaciones de otro tipo a menos que se haga una conversión explícita. Por ejemplo: Java, Pascal y Python.
- **Débilmente tipados:** se pueden mezclar en una misma expresión valores de distinto tipo. Por ejemplo PHP y JavaScript.

```
x = "a" + 5
```

```
In [ ]: x = "a" + 5  
        x
```



Python

- Es un lenguaje **fuertemente tipado**.
- Posee un **tipado dinámico**: el intérprete de Python realiza el chequeo de tipos durante la ejecución y el tipo de una variable puede cambiar durante su tiempo de vida.

La verificación de tipos

- Se refiere a chequeo de tipos.
- Es donde se aplican las reglas definidas en el sistema de tipos.
- La verificación de tipos puede ser:
 - estática: ocurre en tiempo de **compilación**. Por ejemplo: Pascal y C
 - dinámica: ocurre en tiempo de **ejecución**. Por ejemplo PHP, Ruby y Python.

```
In [ ]: option = input("Ingresa 1 para verificar y 2 para no: ")
if option == "1":
    print("Estoy chequeando...")
    print("e" + 4 )
else:
    print("Ahora no estoy dando error")
```

Duck Typing

"Si parece un pato, nada como un pato y suena como un pato, entonces probablemente sea un pato"

Observemos el siguiente código

- Sacado de <https://realpython.com/python-type-checking/>

```
In [ ]: def headline(text, align=True):
        if align:
            return f"{text.title()}\n{'-' * len(text)}"
        else:
            return f" {text.title()} ".center(50, "-")
```

```
In [ ]: print(headline("python type checking"))
```

```
In [ ]: print(headline("python type checking", align=False))
```

Probemos esto:

```
In [ ]: print(headline("python type checking", align="left"))
```

Python permite agregar sugerencias de tipos: anotaciones

```
In [ ]: def headline(text: str, align: bool = True) -> str:
        if align:
            return f"{text.title()}\n{'-' * len(text)}"
        else:
            return f" {text.title()} ".center(50, "-")
```

```
In [ ]: print(headline("python type checking", align="left"))
```

- ¿Cambió algo?

Si bien estas anotaciones están disponibles en tiempo de ejecución a través del atributo `__annotations__`, no se realiza ninguna verificación de tipo en tiempo de ejecución.

```
In [ ]: headline.__annotations__
```

Pero si lo abrimos en un IDE (PyCharm en este caso)

```
main.py x
1 def headline(text: str, align: bool = True) -> str:
2     if align:
3         return f"{text.title()}\n{'-' * len(text)}"
4     else:
5         return f" {text.title()} ".center(50, "-")
6     print(headline("python type checking", align="left"))
7
```

Expected type 'bool', got 'str' instead

- Usamos un verificador de tipos externo.
- La herramienta más común para realizar la verificación de tipos es [Mypy](#)

mypy

- Se instala con pip: `pip install mypy`

```
Terminal: Local x +
claudia@bruce:~/ownCloud/Materias/Python/2021/entorno2021/ejemplos_de_clase/venv$ mypy tipos.py
tipos.py:6: error: Argument "align" to "headline" has incompatible type "str"; expected "bool"
Found 1 error in 1 file (checked 1 source file)
```

¿Cómo resolvemos este "error"?

```
In [ ]: def headline(text: str, centered: bool = True) -> str:
        if centered:
            return f"{text.title()}\n{'-' * len(text)}"
        else:
            return f" {text.title()} ".center(50, "-")
```

```
In [ ]: print(headline("python type checking"))
        print(headline("use mypy", centered=True))
```

Anotaciones

- Como vimos, en las funciones se puede agregar anotaciones sobre los argumentos y el valor de retorno.
- En general:

```
def funcion(arg1: arg_type, arg2: arg_type = valor) -> return_type:
    ...
```

```
In [ ]: import math

def area_circunferencia(radius: float) -> float:
    return math.pi * radius ** 2

area = area_circunferencia(2)
print(area)
```

También se pueden hacer anotaciones de variables

```
In [ ]: pi: float = 3.1415

def area_circunferencia(radio: float) -> float:
    return math.pi * radio ** 2

area = area_circunferencia(2)
print(area)
```

```
In [ ]: area_circunferencia.__annotations__
```

```
In [ ]: __annotations__
```

Un poco más sobre anotaciones

- Se puede realizar una anotación de una variable **sin darle un valor**.

```
In [ ]: message: str
        __annotations__
```

```
In [ ]: message = 10
        message
```

Otros ejemplos

```
In [ ]: rock_bands: list = ["Led Zeppelin", "AC/DC", "Queen"]
        notes: tuple = (7, 8, 9, 10)
        options: dict = {"centered": False, "capitalize": True}
```

- ¿Cómo podemos indicar que se trata de una lista de elementos str? ¿O una tupla de enteros?

El modulo typing

- Permite escribir anotaciones un poco más complejas.

```
In [ ]: from typing import Dict, List, Tuple

        rock_bands: List[str] = ["Led Zeppelin", "AC/DC", "Queen"]
        notes: Tuple[int, int, int, int] = (7, 8, 9, 10)
        options: Dict[str, bool] = {"centered": False, "capitalize": True}
```

Veamos este otro ejemplo

```
In [ ]: from typing import List, Sequence

        def squares(elms: Sequence[float]) -> List[float]:
            return [x**2 for x in elems]
```

```
In [ ]: squares([1, 2, 3])
```

- Una secuencia es cualquier objeto que admita `len ()` y `__getitem__ ()`, independientemente de su tipo real.

¿Qué pasa con este código?

```
In [ ]: import random

def random_element(list_of_items):
    return random.choice(list_of_items)

my_list = [1, "dos", 3.1415]
random_element(my_list)
```

Para incorporar las anotaciones usamos el tipo: Any

```
In [ ]: import random
from typing import Any, Sequence

def random_element(list_of_items: Sequence[Any]) -> Any:
    return random.choice(list_of_items)

my_list = [1, "dos", 3.1415]
random_element(my_list)
```

Anotaciones y POO

¿Cómo agregamos anotaciones a los métodos?

```
In [ ]: class Jugador:

    def __init__(self,
                 nombre: str,
                 juego: str = "Tetris",
                 tiene_equipo: bool = False,
                 equipo: str = None) -> None:

        self.nombre = nombre
        self.juego = juego
        self.tiene_equipo = tiene_equipo
        self.equipo = equipo

    def jugar(self) -> None:
        if self.tiene_equipo:
            print(f"{self.nombre} juega en el equipo {self.equipo} al {self.juego}")
        else:
            print(f"{self.nombre} juega solo al {self.juego}")
```

- Se usan las mismas reglas que para las funciones.
- **self** no necesita ser anotado. ¿De qué tipo es?

¿Cómo agregamos anotaciones a las variables de instancia y de clase?

- Se usan las mismas reglas que para las variables comunes.

```
In [ ]: class SuperHeroe():  
    """ Esta clase define a un superheroe  
    villanos: representa a los enemigos de todos los superhéroes  
    """  
    villanos: List[str] = []  
  
    def __init__(self, nombre: str, alias: str) -> None:  
        self._nombre = nombre  
        self._enemigos = []
```

Hasta acá llegamos...

Más info

- La [PEP 3107](#) introdujo la sintaxis para las anotaciones de funciones, pero la semántica se dejó deliberadamente sin definir.
- La [PEP 484](#) introduce un módulo provisional para proporcionar definiciones y herramientas estándares, junto con algunas convenciones para situaciones en las que las anotaciones no están disponibles.
- La [PEP 526](#): tiene como objetivo mostrar de qué manera se pueden relajar anotaciones de variables (incluidas las variables de clase y las variables de instancia),
- Artículo de RealPython: <https://realpython.com/python-type-checking/>
- Artículo de [the state of type hints in Python](#) de Bernát Gábor.

¡Nos vemos el martes!



Facultad de
INFORMÁTICA
UNIVERSIDAD NACIONAL DE LA PLATA



UNIVERSIDAD
NACIONAL
DE LA PLATA