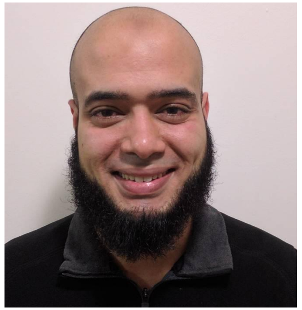
Python Programming

Function and Variable Type **Annotations**

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

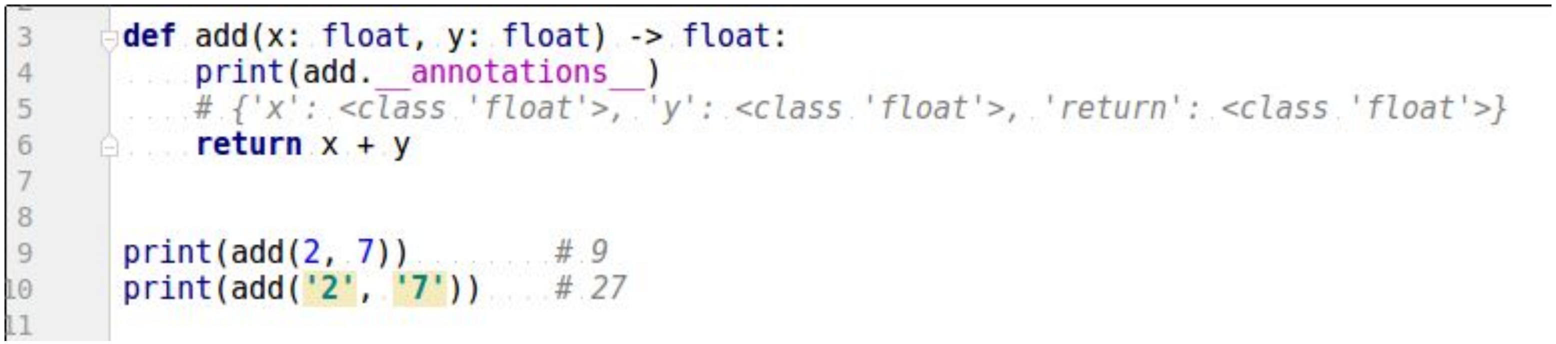


PEP

- **PEP** stands for Python Enhancement Proposal: a design document providing information to the Python community, or describing a new feature for Python or its processes or environment.
- **PEP 8**: Style Guide for Python Code
    - E.g. too many blank lines
- **Function Annotations** – PEP 3107 / **Type Hints** - PEP 484

Function Annotations

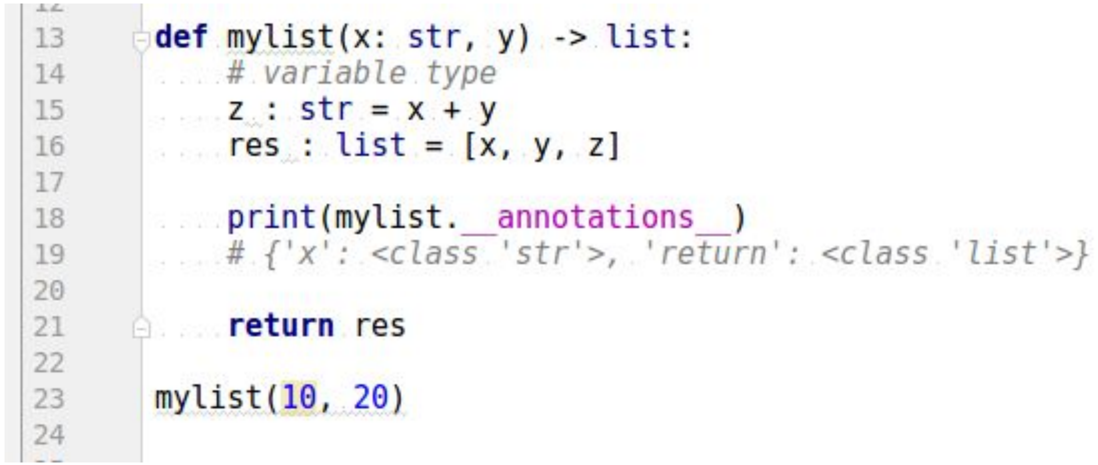
- We can state the expected data type for the arguments and return
- However, python interpreter just discard them!
  - But they still can communicate for the user what type of arguments to pass!
  - Also, some third library static type checker (e.g. mypy) can be applied before running code!



```python
def add(x: float, y: float) -> float:
    print(add.__annotations__)
    # {'x': <class 'float'>, 'y': <class 'float'>, 'return': <class 'float'>}
    return x + y


print(add(2, 7))        # 9
print(add('2', '7'))    # 27
```

Type Hints

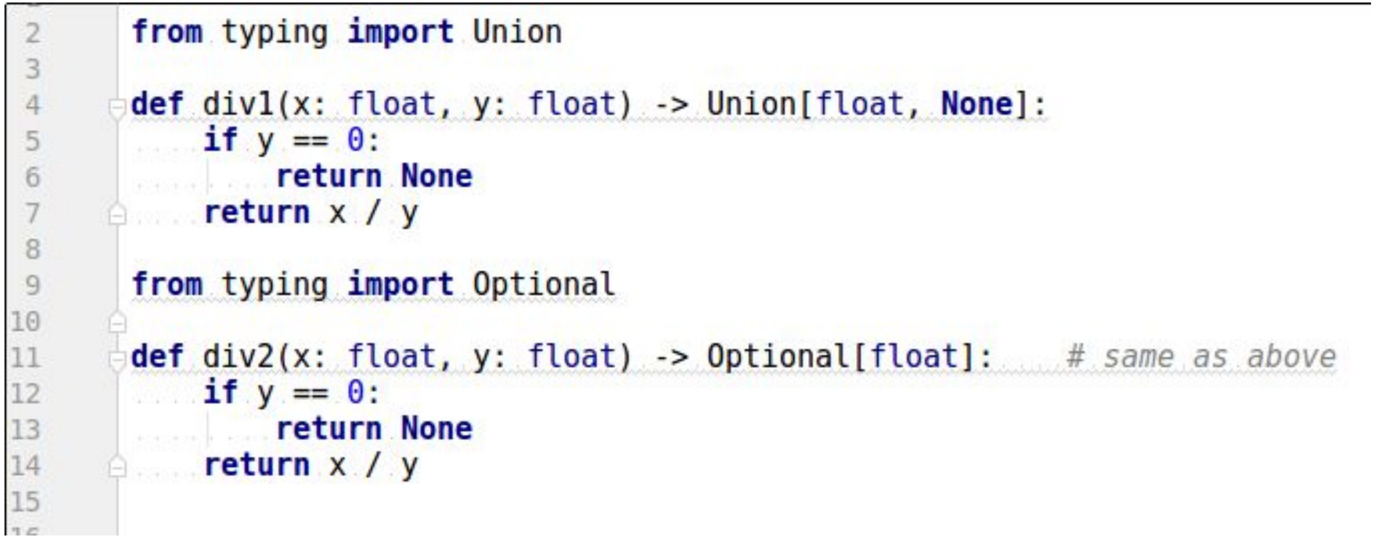
- We can even state the expected data type for the variables!



```
def mylist(x: str, y) -> list:
    # variable type
    z : str = x + y
    res : list = [x, y, z]

    print(mylist.__annotations__)
    # {'x': <class 'str'>, 'return': <class 'list'>}

    return res

mylist(10, 20)
```

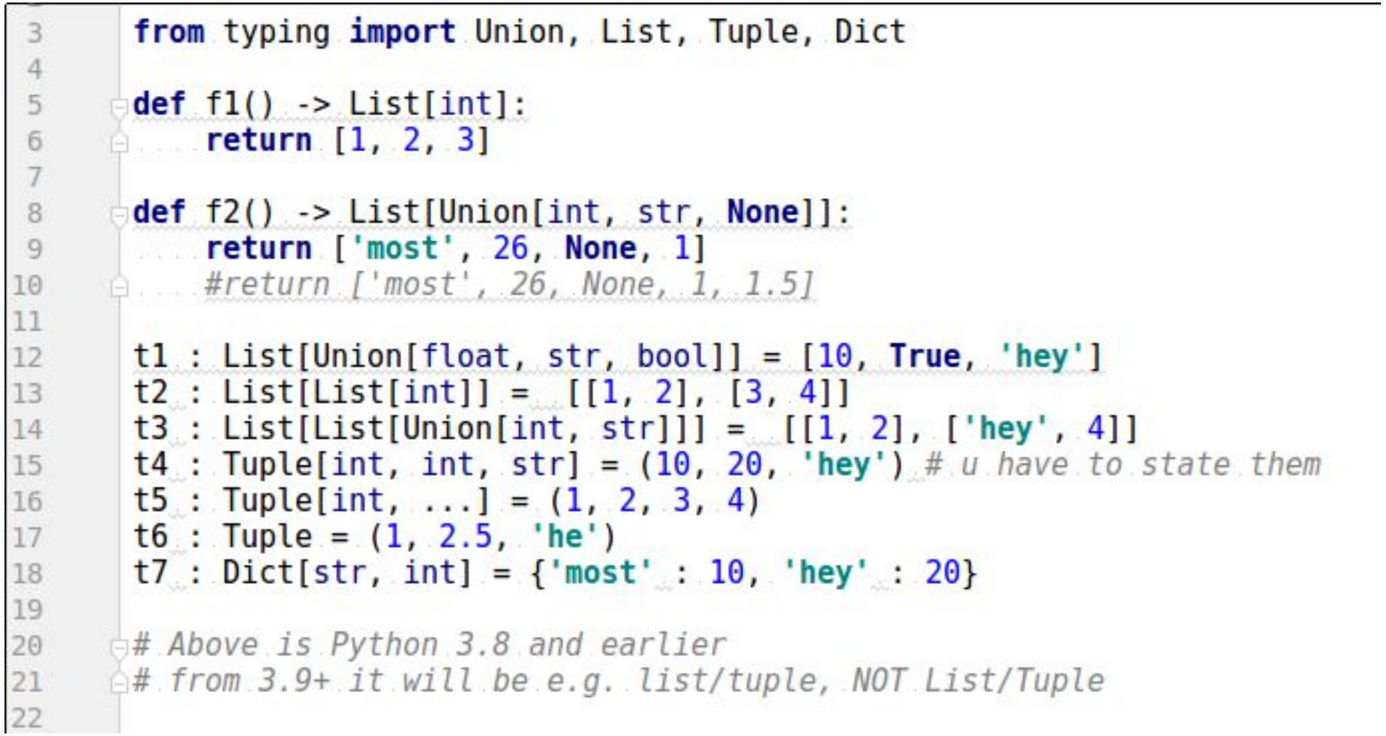
Complex typing

- What if I would like to return something that could be 2+ data types?
  - Use Union to indicate them
  - Optional[] means can be None



```python
from typing import Union

def div1(x: float, y: float) -> Union[float, None]:
    if y == 0:
        return None
    return x / y

from typing import Optional

def div2(x: float, y: float) -> Optional[float]:    # same as above
    if y == 0:
        return None
    return x / y

```

Complex typing: More



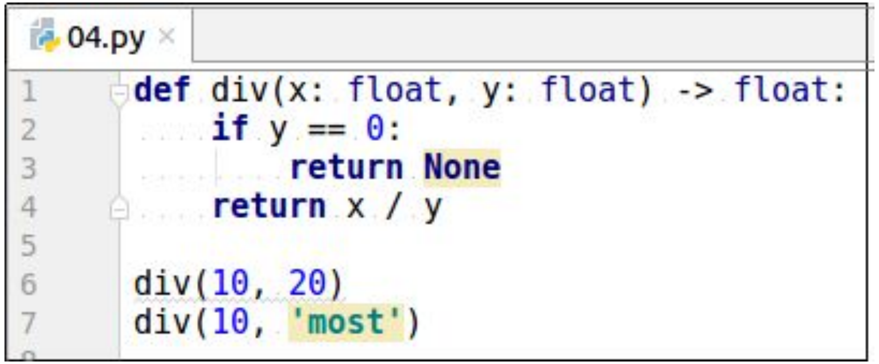
```python
from typing import Union, List, Tuple, Dict

def f1() -> List[int]:
    return [1, 2, 3]

def f2() -> List[Union[int, str, None]]:
    return ['most', 26, None, 1]
    #return ['most', 26, None, 1, 1.5]

t1 : List[Union[float, str, bool]] = [10, True, 'hey']
t2 : List[List[int]] =  [[1, 2], [3, 4]]
t3 : List[List[Union[int, str]]] =  [[1, 2], ['hey', 4]]
t4 : Tuple[int, int, str] = (10, 20, 'hey') # u have to state them
t5 : Tuple[int, ...] = (1, 2, 3, 4)
t6 : Tuple = (1, 2.5, 'he')
t7 : Dict[str, int] = {'most' : 10, 'hey' : 20}

# Above is Python 3.8 and earlier
# from 3.9+ it will be e.g. list/tuple, NOT List/Tuple
```

Overall

- Python interpreter **doesn’t consider them**
    - You expect int, but still float or string work well (e.g. a + b)
- Using them may encourage you think about your I/O & code logic
- Communicate clearly the arguments & return, especially for APIs
- There are **3rd party tools** to statically check your code (mypy)
- There is a debate around using them: Is it pythonic?
    - Replacement: Duck typing + try-except block
- Think: is it added value to use? If yes, use them wisely
- Future Reading

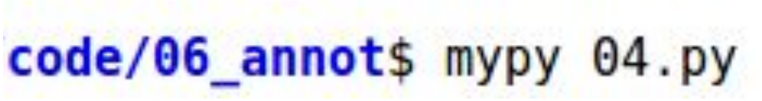
mypy tool

- In some companies, some 3rd libraries can be used to statically check
  - Popular one such as pymy: **pip3 install mypy**

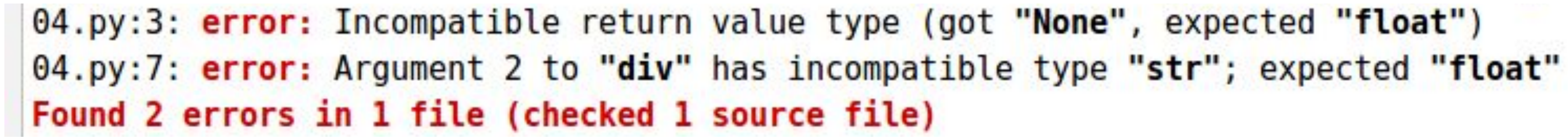


04.py

```python
def div(x: float, y: float) -> float:
    if y == 0:
        return None
    return x / y

div(10, 20)
div(10, 'most')
```



```
code/06_annot$ mypy 04.py
```



```
04.py:3: error: Incompatible return value type (got "None", expected "float")
04.py:7: error: Argument 2 to "div" has incompatible type "str"; expected "float"
Found 2 errors in 1 file (checked 1 source file)
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."