(/)

Curriculum
Short Specializations
Average: **97.3%**

0x00. Python - Variable Annotations

Python **Back-end**

By: Emmanuel Turlay, Staff Software Engineer at Cruise

Weight: 1

Project over - took place from Dec 7, 2023 6:00 AM to Dec 8, 2023 6:00 AM

An auto review will be launched at the deadline

In a nutshell…

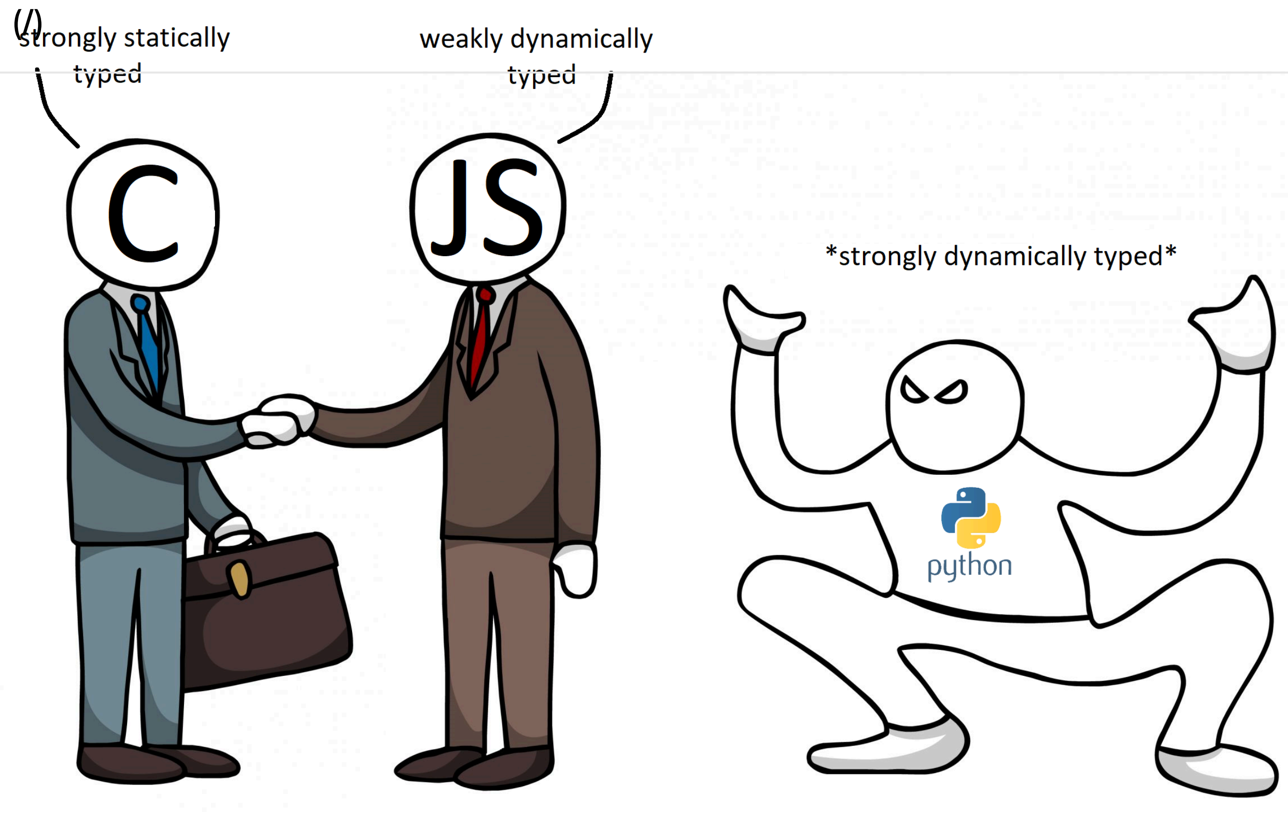
- **Auto QA review:** 53.0/53 mandatory & 5.0/10 optional
- **Altogether: 150.0%**
  - Mandatory: 100.0%
  - Optional: 50.0%
  - Calculation: 100.0% + (100.0% * 50.0%) == **150.0%**

Concepts

For this project, we expect you to look at this concept:

- Advanced Python (/concepts/554)

(/)



Resources

Read or watch:

- Python 3 typing documentation (/rltoken/5j0OtdWh36_HVAHKJX2gaA)
- MyPy cheat sheet (/rltoken/Eud-nrUG7x3iT6JD2Sas-g)

Learning Objectives

General

At the end of this project, you are expected to be able to explain to anyone (/rltoken/hGUom4nCewYmroS4ii_ZDQ), **without the help of Google**:

- Type annotations in Python 3
- How you can use type annotations to specify function signatures and variable types
- Duck typing
- How to validate your code with `mypy`

Requirements

General

- Allowed editors: `vi`, `vim`, `emacs`
- All your files will be interpreted/compiled on Ubuntu 18.04 LTS using `python3` (version 3.7)
- All your files should end with a new line

(/)

- The first line of all your files should be exactly `#!/usr/bin/env python3`
- A `README.md` file, at the root of the folder of the project, is mandatory
- Your code should use the `pycodestyle` style (version 2.5.)
- All your files must be executable
- The length of your files will be tested using `wc`
- All your modules should have a documentation (`python3 -c 'print(__import__("my_module").__doc__)'`)
- All your classes should have a documentation (`python3 -c 'print(__import__("my_module").MyClass.__doc__)'`)
- All your functions (inside and outside a class) should have a documentation (`python3 -c 'print(__import__("my_module").my_function.__doc__)'` and `python3 -c 'print(__import__("my_module").MyClass.my_function.__doc__)'`)
- A documentation is not a simple word, it's a real sentence explaining what's the purpose of the module, class or method (the length of it will be verified)

Tasks

0. Basic annotations - add

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a type-annotated function `add` that takes a float `a` and a float `b` as arguments and returns their sum as a float.

```
bob@dylan:~$ cat 0-main.py
#!/usr/bin/env python3
add = __import__('0-add').add

print(add(1.11, 2.22) == 1.11 + 2.22)
print(add.__annotations__)

bob@dylan:~$ ./0-main.py
True
{'a':  <class 'float'>, 'b': <class 'float'>, 'return': <class 'float'>}
```

Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x00-python_variable_annotations`
- File: `0-add.py`

Done! Help Check your code Get a sandbox QA Review

1. Basic annotations - concat

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a type-annotated function `concat` that takes a string `str1` and a string `str2` as arguments and returns a concatenated string

```
bob@dylan:~$ cat 1-main.py
#!/usr/bin/env python3
concat = __import__('1-concat').concat

str1 = "egg"
str2 = "shell"

print(concat(str1, str2) == "{}{}".format(str1, str2))
print(concat.__annotations__)

bob@dylan:~$ ./1-main.py
True
{'str1': <class 'str'>, 'str2': <class 'str'>, 'return': <class 'str'>}
```

Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x00-python_variable_annotations`
- File: `1-concat.py`

Done! Help Check your code Get a sandbox QA Review

2. Basic annotations - floor

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a type-annotated function `floor` which takes a float `n` as argument and returns the floor of the float.

```
bob@dylan:~$ cat 2-main.py
#!/usr/bin/env python3

import math

floor = __import__('2-floor').floor

ans = floor(3.14)

print(ans == math.floor(3.14))
print(floor.__annotations__)
print("floor(3.14) returns {}, which is a {}".format(ans, type(ans)))

bob@dylan:~$ ./2-main.py
True
{'n': <class 'float'>, 'return': <class 'int'>}
floor(3.14) returns 3, which is a <class 'int'>
```

Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x00-python_variable_annotations`
- File: `2-floor.py`

Done! Help Check your code Get a sandbox QA Review

3. Basic annotations - to string

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a type-annotated function `to_str` that takes a float `n` as argument and returns the string representation of the float.

```
bob@dylan:~$ cat 3-main.py
#!/usr/bin/env python3
to_str = __import__('3-to_str').to_str

pi_str = to_str(3.14)
print(pi_str == str(3.14))
print(to_str.__annotations__)
print("to_str(3.14) returns {} which is a {}".format(pi_str, type(pi_str)))

bob@dylan:~$ ./3-main.py
True
{'n': <class 'float'>, 'return': <class 'str'>}
to_str(3.14) returns 3.14, which is a <class 'str'>
```

(/)
Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x00-python_variable_annotations`
- File: `3-to_str.py`

Done! Help Check your code Get a sandbox QA Review

4. Define variables

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Define and annotate the following variables with the specified values:

- `a` , an integer with a value of 1
- `pi` , a float with a value of 3.14
- `i_understand_annotations` , a boolean with a value of True
- `school` , a string with a value of “Holberton”

```
bob@dylan:~$ cat 4-main.py
#!/usr/bin/env python3

a = __import__('4-define_variables').a
pi = __import__('4-define_variables').pi
i_understand_annotations = __import__('4-define_variables').i_understand_annotations
school = __import__('4-define_variables').school

print("a is a {} with a value of {}".format(type(a), a))
print("pi is a {} with a value of {}".format(type(pi), pi))
print("i_understand_annotations is a {} with a value of {}".format(type(i_understand
_annotations), i_understand_annotations))
print("school is a {} with a value of {}".format(type(school), school))

bob@dylan:~$ ./4-main.py
a is a <class 'int'> with a value of 1
pi is a <class 'float'> with a value of 3.14
i_understand_annotations is a <class 'bool'> with a value of True
school is a <class 'str'> with a value of Holberton
```

Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x00-python_variable_annotations`
- File: `4-define_variables.py`

Done! Help Check your code Get a sandbox QA Review

5. Complex types - list of floats

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a type-annotated function `sum_list` which takes a list `input_list` of floats as argument and returns their sum as a float.

```
bob@dylan:~$ cat 5-main.py
#!/usr/bin/env python3

sum_list = __import__('5-sum_list').sum_list

floats = [3.14, 1.11, 2.22]
floats_sum = sum_list(floats)
print(floats_sum == sum(floats))
print(sum_list.__annotations__)
print("sum_list(floats) returns {} which is a {}".format(floats_sum, type(floats_su
m)))

bob@dylan:~$ ./5-main.py
True
{'input_list': typing.List[float], 'return': <class 'float'>}
sum_list(floats) returns 6.470000000000001 which is a <class 'float'>
```

Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x00-python_variable_annotations`
- File: `5-sum_list.py`

Done! Help Check your code Get a sandbox QA Review

6. Complex types - mixed list

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a type-annotated function `sum_mixed_list` which takes a list `mxd_lst` of integers and floats and returns their sum as a float.

```
bob@dylan:~$ cat 6-main.py
#!/usr/bin/env python3

sum_mixed_list = __import__('6-sum_mixed_list').sum_mixed_list

print(sum_mixed_list.__annotations__)
mixed = [5, 4, 3.14, 666, 0.99]
ans = sum_mixed_list(mixed)
print(ans == sum(mixed))
print("sum_mixed_list(mixed) returns {} which is a {}".format(ans, type(ans)))

bob@dylan:~$ ./6-main.py
{'mxd_lst': typing.List[typing.Union[int, float]], 'return': <class 'float'>}
True
sum_mixed_list(mixed) returns 679.13 which is a <class 'float'>
```

Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x00-python_variable_annotations`
- File: `6-sum_mixed_list.py`

Done! Help Check your code Get a sandbox QA Review

7. Complex types - string and int/float to tuple

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a type-annotated function `to_kv` that takes a string `k` and an int OR float `v` as arguments and returns a tuple. The first element of the tuple is the string `k`. The second element is the square of the int/float `v` and should be annotated as a float.

```
bob@dylan:~$ cat 7-main.py
#!/usr/bin/env python3

to_kv = __import__('7-to_kv').to_kv

print(to_kv.__annotations__)
print(to_kv("eggs", 3))
print(to_kv("school", 0.02))

bob@dylan:~$ ./7-main.py
{'k': <class 'str'>, 'v': typing.Union[int, float], 'return': typing.Tuple[str, floa
t]}
('eggs', 9)
('school', 0.0004)
```

(/)
Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x00-python_variable_annotations`
- File: `7-to_kv.py`

Done! Help Check your code Get a sandbox QA Review

8. Complex types - functions

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a type-annotated function `make_multiplier` that takes a float `multiplier` as argument and returns a function that multiplies a float by `multiplier`.

```
bob@dylan:~$ cat 8-main.py
#!/usr/bin/env python3

make_multiplier = __import__('8-make_multiplier').make_multiplier
print(make_multiplier.__annotations__)
fun = make_multiplier(2.22)
print("{}".format(fun(2.22)))

bob@dylan:~$ ./8-main.py
{'multiplier': <class 'float'>, 'return': typing.Callable[[float], float]}
4.928400000000001
```

Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x00-python_variable_annotations`
- File: `8-make_multiplier.py`

Done! Help Check your code Get a sandbox QA Review

9. Let's duck type an iterable object

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Annotate the below function's parameters and return values with the appropriate types

```
def element_length(lst):
    return [(i, len(i)) for i in lst]
```

```
bob@dylan:~$ cat 9-main.py
#!/usr/bin/env python3

element_length =  __import__('9-element_length').element_length

print(element_length.__annotations__)

bob@dylan:~$ ./9-main.py
{'lst': typing.Iterable[typing.Sequence], 'return': typing.List[typing.Tuple[typing.
Sequence, int]]}
```

Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x00-python_variable_annotations`
- File: `9-element_length.py`

☑ Done! Help Check your code >_ Get a sandbox QA Review

10. Duck typing - first element of a sequence

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Augment the following code with the correct duck-typed annotations:

```
# The types of the elements of the input are not know
def safe_first_element(lst):
    if lst:
        return lst[0]
    else:
        return None
```

```
bob@dylan:~$ cat 100-main.py
#!/usr/bin/env python3

safe_first_element =  __import__('100-safe_first_element').safe_first_element

print(safe_first_element.__annotations__)

bob@dylan:~$ ./100-main.py
{'lst': typing.Sequence[typing.Any], 'return': typing.Union[typing.Any, NoneType]}
```

Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x00-python_variable_annotations`
- File: `100-safe_first_element.py`

Done! Help Check your code Get a sandbox QA Review

11. More involved type annotations

#advanced

Score: 66.67% (*Checks completed: 66.67%*)

Given the parameters and the return values, add type annotations to the function

Hint: look into TypeVar

```
def safely_get_value(dct, key, default = None):
    if key in dct:
        return dct[key]
    else:
        return default
```

```
bob@dylan:~$ cat 101-main.py
#!/usr/bin/env python3

safely_get_value = __import__('101-safely_get_value').safely_get_value
annotations = safely_get_value.__annotations__

print("Here's what the mappings should look like")
for k, v in annotations.items():
    print( ("{}: {}".format(k, v)))

bob@dylan:~$ ./101-main.py
Here's what the mappings should look like
dct: typing.Mapping
key: typing.Any
default: typing.Union[~T, NoneType]
return: typing.Union[typing.Any, ~T]
```

Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x00-python_variable_annotations`
- File: `101-safely_get_value.py`

(/)Done? Help Check your code Ask for a new correction >_ Get a sandbox QA Review

12. Type Checking

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Use `mypy` to validate the following piece of code and apply any necessary changes.

```
def zoom_array(lst: Tuple, factor: int = 2) -> Tuple:
    zoomed_in: Tuple = [
        item for item in lst
        for i in range(factor)
    ]
    return zoomed_in


array = [12, 72, 91]

zoom_2x = zoom_array(array)

zoom_3x = zoom_array(array, 3.0)
```

```
bob@dylan:~$ mypy 102-type_checking.py
Success: no issues found in 1 source file
bob@dylan:~$ cat 102-main.py
#!/usr/bin/env python3

zoom_array =  __import__('102-type_checking').zoom_array

print(zoom_array.__annotations__)

bob@dylan:~$ ./102-main.py
{'lst': typing.Tuple, 'factor': <class 'int'>, 'return': typing.List}
```

Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x00-python_variable_annotations`
- File: `102-type_checking.py`

Done? Help Check your code Ask for a new correction >_ Get a sandbox QA Review

(/)