MyPy

How to make your buggy simulation crash now instead of after 24 hours of computation

Denis Mérigoux

Prosecco – Inria

Fev 19th 2019

The billion dollar mistake

Suppose you are launching a 7-days long deep neural network training on the cluster. The deadline for NIPS is in 10 days, so you should have plenty of time to evaluate your newly-trained model and put the final touch on your paper.

The billion dollar mistake

Suppose you are launching a 7-days long deep neural network training on the cluster. The deadline for NIPS is in 10 days, so you should have plenty of time to evaluate your newly-trained model and put the final touch on your paper.

Everything is going fine for the first 6 days, your excitement grows as your program prints completion statistics on the standard output.

The billion dollar mistake

Suppose you are launching a 7-days long deep neural network training on the cluster. The deadline for NIPS is in 10 days, so you should have plenty of time to evaluate your newly-trained model and put the final touch on your paper.

Everything is going fine for the first 6 days, your excitement grows as your program prints completion statistics on the standard output.

However, at 3am on Saturday night, just before the programs saves the trained model to memory, the dreadful message appears on your `tmux` session :

```
TypeError: unsupported
  operand type(s) for +:
  'NoneType' and 'int'
```

Escaping the hellish fatality of dynamic type errors

As you enter the third stage of grief, you start bargaining with your poor programming practices :

— *I should have tested on small examples first!*
— *How could I forget to check if this counter was* `None`*?*
— *Maybe I can buy some AWS credits and make this training run in 3 days for the deadline!*

Escaping the hellish fatality of dynamic type errors

As you enter the third stage of grief, you start bargaining with your poor programming practices :

> — *I should have tested on small examples first!*
> — *How could I forget to check if this counter was* `None`*?*
> — *Maybe I can buy some AWS credits and make this training run in 3 days for the deadline!*

However, this reflexion usually ends up with a painful self-blame coupled with firm resolutions about being more careful next time.

Escaping the hellish fatality of dynamic type errors

As you enter the third stage of grief, you start bargaining with your poor programming practices :

> — *I should have tested on small examples first!*
> — *How could I forget to check if this counter was* `None`*?*
> — *Maybe I can buy some AWS credits and make this training run in 3 days for the deadline!*

However, this reflexion usually ends up with a painful self-blame coupled with firm resolutions about being more careful next time.

What if this was a tooling problem
rather than a programming practices problem?

But how to check before execution ?

By giving more information about the values of your program!

```
# Annotations supported from Python 3.5!
d: Dict[str, int] = { "foo": 0 }
def get_val(s: str) -> int:
  return d[s]
```

How is that useful?

The `mypy` tool will read the annotations and check, *without executing the program*, whether everything is consistent.

Will it slow things down?

Checking is very fast (< 1 s usually), and annotations are ignored during actual execution.

The typing miracle

```
1  def activation_function(x: float) -> Optional[float]:
2      return arctan(x)
3
4  def perceptron_result(sum: float) -> float:
5      res = activation_function(sum)
6      return res
```

```
> mypy myprogram.py
myprogram.py:6: error: Incompatible return value type
(got "Optional[float]", expected "float")
```

The typing miracle

```
1  def activation_function(x: float) -> Optional[float]:
2      return arctan(x)
3
4  def perceptron_result(sum: float) -> float:
5      res = activation_function(sum)
6      if res is None:
7          raise ExceptionYouCanCatch
8      else:
9          return res
```

Machine-assisted programming practices improvement

Take advantage of modern tooling to eliminate the mistakes that keep dragging you down!

- Use Python >3.5 and the typing module to start annotating your code.
- Use Mypy to check before execution.
- Benefit from more code that is both less buggy and more understandable.