Using types to rule out bugs: mypy worksheet

Dominic Orchard

This short worksheet will take you through a few exercises to practice using mypy. Here is an example as a reminder of the syntax for inserting type specifications:

```
def myAbs(x : float) -> float:
  """Take the absolute of the floating-point input"""
  if x < 0:
    return (-x)
  else:
    return x
```

You can find the standard documentation for Python's `typing` library at:

`https://docs.python.org/3/library/typing.html`

1. Define a function `mean` to take the mean of two floats, giving its full type specification.

   Check that `mypy` accepts this (i.e., run `mypy yourfile.py`).

2. Experiment with modifying your function to something that would produce a runtime type error and check how `mypy` reports this.

3. Using `reveal_type` find out what the type of the `len` function is according to `mypy`.

   (You might like to guard this with a conditional checking the `TYPE_CHECKING` value so you can leave it in and run your code later).

Consider the following Python code to calculate the mean of a list:

```
def meanN(xs : list[float]) -> float:
  sum = 0.0
  for x in xs:
    sum+=x
  return (sum / len(xs))
```

There is nothing specific in this code to lists; it can be generalised to arbitrary iterable data types. Note however, that we use `len` which requires a `Sized` value, and iteration requires `Iterable`. Instead we can use the `Collection` class from the `typing` module (which inherits from both `Iterable` and `Sized`).

4. Import the `Collection` class and modify the above to work on arbitrary collections.

5. Define a generalisation of `meanN` that takes an additional function parameter. Instead of computing the mean directly, this new function `meanGen` should sum up all the elements of the collection then apply the parameter function to this value to produce the output.

   Give it a type signature with the most general type.

   Hint: Use the `Callable` class. You can introduce a parametric type variable via: `T = TypeVar('T')`

   Write some code to use this to compute the mean of some test data.

6. (**Challenge**) We can further generalise `meanGen` into a function `reducer` which captures the general pattern of reducing a collection into a single value, taking as inputs:

   - A collection of some arbitrary element type `T`;
   - A function with two inputs for combining `T` values with a partially reduced value `S`, to produce a new reduced value `S`;
   - An initial reduced value `S`;
   - A final transformation function mapping from the reduced value `S` to the final result of type `U`.

   Implement this general reducer and redefine the mean of a collection in terms of it, giving a simple test.

Solutions: `http://dorchard.github.io/types-tutorial/mypy-worksheet-solutions.py`