Generic Types and Filter Algorithm

Today starts as a Paper + Pencil or Tablet + Pencil day... please keep laptops stowed away!

COMP110 - CL16

2024/04/02

Warm-up: Fill in the blanks with valid function call names.

This will be today's attendance submission (CL16). Submit once complete.

```python
from typing import Callable


def map(f: Callable[[str], int], xs: list[str]) -> list[int]:
    result: list[int] = []
    for x in xs:
        result.append(f(x))
    return result


def concat(x: str) -> str:
    return x + x


def length(x: str) -> int:
    return len(x)


values: list[str] = ["a", "bb", "ccc"]
mapped: list[___A___] = map(___B___, values)

print(mapped)  # C
```

Warm-up #2: Fill in the blanks A-E to the right

```python
def map1(f: Callable[[int], int], xs: list[int]) -> list[int]:
    result: list[int] = []
    for x in xs:
        result.append(f(x))
    return result


def map2(f: Callable[[str], int], xs: list[str]) -> list[int]:
    result: list[int] = []
    for x in xs:
        result.append(f(x))
    return result


def length(s: str) -> int:
    return int(s)


def double(x: int) -> str:
    return x * 2
```

```python
strs: list[str] = ["10", "20", "30"]
ints: list[int] = ___A___(___B___, strs)
results: list[int] = ___C___(___D___, ints)
print(results)  # ____E____ Expected output
```

What's unsatisfying about these two functions?

```python
def map1(f: Callable[[int], int], xs: list[int]) -> list[int]:
    result: list[int] = []
    for x in xs:
        result.append(f(x))
    return result
```

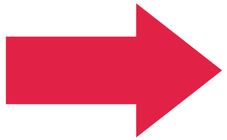
```python
def map2(f: Callable[[str], int], xs: list[str]) -> list[int]:
    result: list[int] = []
    for x in xs:
        result.append(f(x))
    return result
```

Introducing TypeVar and Generics

from typing import TypeVar

Example:

```python
def map1(f: Callable[[int], int], xs: list[int]) -> list[int]:
    result: list[int] = []
    for x in xs:
        result.append(f(x))
    return result
```



```python
from typing import Callable, TypeVar

T = TypeVar("T")


def map(f: Callable[[T], T], xs: list[T]) -> list[T]:
    result: list[T] = []
    for x in xs:
        result.append(f(x))
    return result
```

Example Usage

```python
from typing import Callable, TypeVar

T = TypeVar("T")


def map(f: Callable[[T], T], xs: list[T]) -> list[T]:
    result: list[T] = []
    for x in xs:
        result.append(f(x))
    return result
```

```python
def double(x: int) -> int:
    return x * 2


values: list[int] = [2, 4]
ys: list[int] = map(double, values)
print(ys)
```

```python
def excite(s: str) -> str:
    return s + "!"


letters: list[str] = ["U", "N", "C"]
excited: list[str] = map(excite, letters)
print(excited)
```

Discuss: If, *per call,* T can only be *one type,* is the *call* to the given *map* def valid?

(The *len* function is the built-in you know well!)

```
 6    def map(f: Callable[[T], T], xs: list[T]) -> list[T]:
 7        result: list[T] = []
 8        for x in xs:
 9            result.append(f(x))
10        return result
```

```
21    names: list[str] = ["a", "bb", "ccc"]
22    lens: list[int] = map(len, names)
23    print(lens)
```

Hint: what is the Callable type of *len* when given a *str* as an argument?

Multiple TypeVars

What are the *concrete types of* T and U for each of the given three calls to map (on the right)?

```python
from typing import Callable, TypeVar

T = TypeVar("T")
U = TypeVar("U")


def map(f: Callable[[T], U], xs: list[T]) -> list[U]:
    result: list[U] = []
    for x in xs:
        result.append(f(x))
    return result
```

1

```python
def to_int(s: str) -> int:
    return int(s)

numbers: list[str] = ["1", "2", "3"]
ints: list[int] = map(to_int, numbers)
print(ints)
```

T U

2

```python
def double(x: int) -> int:
    return x * 2

values: list[int] = [2, 4]
ys: list[int] = map(double, values)
print(ys)
```

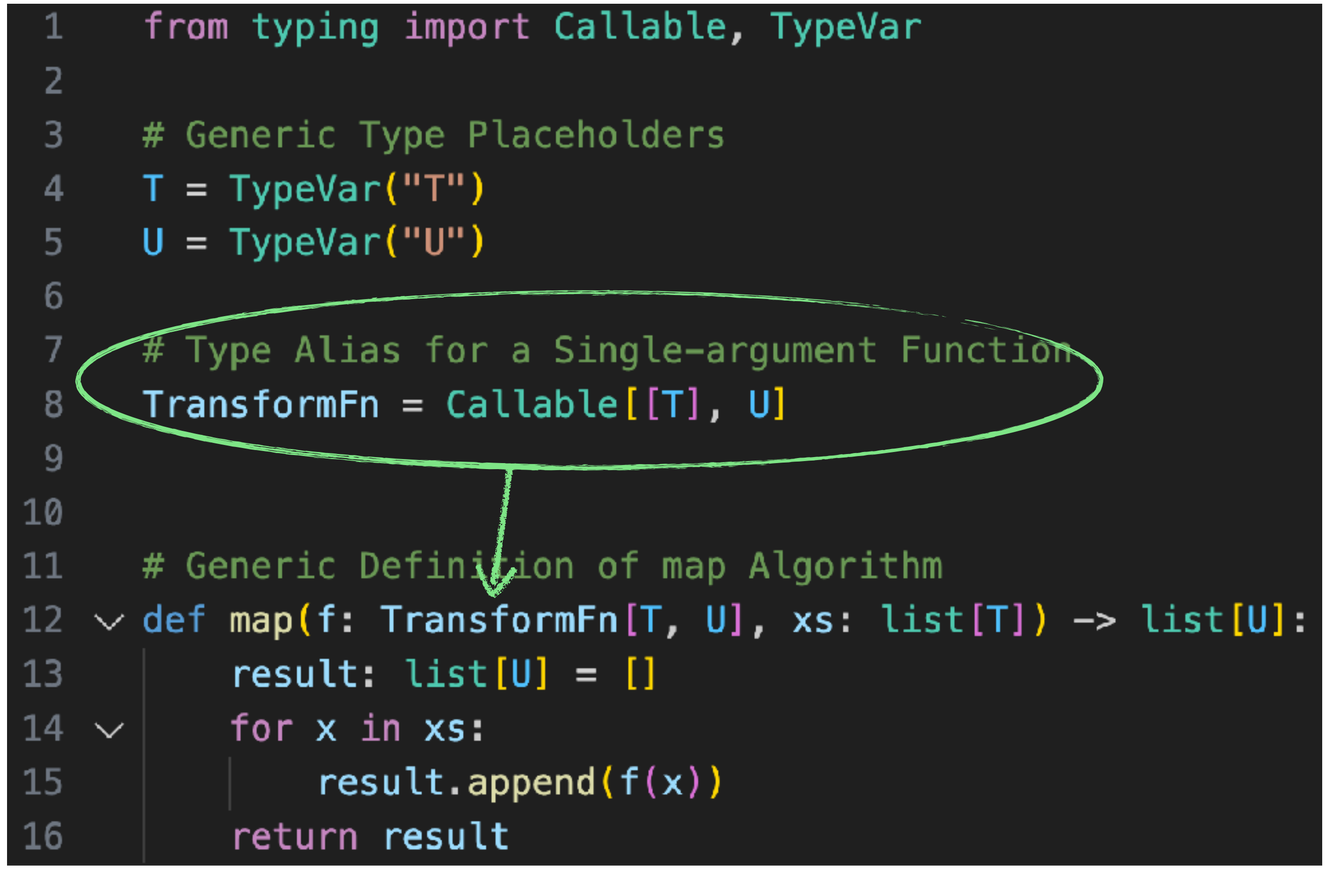
3

```python
def excite(s: str) -> str:
    return s + "!"

letters: list[str] = ["U", "N", "C"]
excited: list[str] = map(excite, letters)
print(excited)
```

Type Aliasing

Giving the *type* of a function a name...



```python
from typing import Callable, TypeVar

# Generic Type Placeholders
T = TypeVar("T")
U = TypeVar("U")

# Type Alias for a Single-argument Function
TransformFn = Callable[[T], U]


# Generic Definition of map Algorithm
def map(f: TransformFn[T, U], xs: list[T]) -> list[U]:
    result: list[U] = []
    for x in xs:
        result.append(f(x))
    return result
```

Function Writing Practice

- Start a new file in your lecture directory, add directory named cl16, create a Python module named filter.py
- Import Callable -- **from typing import Callable**
- Write two functions:
  - A function named *is_odd* that takes an int and returns a bool
  - A function named *filter* that takes a ***Callable[[int], bool]*** and ***list of ints***. For each input int, it should call the callable and if the callable returns True, then add it to a result list. Finally, return the result list.
- Try calling each function in the Trailhead REPL to confirm correctness

Solution

```python
from typing import Callable


def is_odd(x: int) -> bool:
    return x % 2 == 1


def filter(fn: Callable[[int], bool], xs: list[int]) -> list[int]:
    results: list[int] = []
    for x in xs:
        if fn(x):
            results.append(x)
    return results
```

Goal

```python
from typing import Callable, TypeVar

T = TypeVar("T")

Predicate = Callable[[T], bool]


def filter(fn: Predicate[T], xs: list[T]) -> list[T]:
    results: list[T] = []
    for x in xs:
        if fn(x):
            results.append(x)
    return results
```

Diagram

```
 8    def filter(fn: Predicate[T], xs: list[T]) -> list[T]:
 9        results: list[T] = []
10        for x in xs:
11            if fn(x):
12                results.append(x)
13        return results
14
15
16    def positive(x: int) -> bool:
17        return x > 0
18
19
20    values: list[int] = [0, -1, 1]
21    positives: list[int] = filter(positive, values)
22    print(positives)
```

```python
def filter(fn: Predicate[T], xs: list[T]) -> list[T]:
    results: list[T] = []
    for x in xs:
        if fn(x):
            results.append(x)
    return results


def positive(x: int) -> bool:
    return x > 0


values: list[int] = [0, -1, 1]
positives: list[int] = filter(positive, values)
print(positives)
```

Using the Built-in *map* and *filter* Functions