Computer Systems Modeling and Verification (USEEN1)

Higher order functions

Exercises taken from the *Python Programming Primer*[1] (and updated with type hints).

Textbook *Introduction to Python for Computational Science and Engineering* (2022), Chapters 7-8.

Corresponding lecture slides from *Computational Science and Engineering in Python*:

- Higher Order Functions
- Higher Order Functions 2: Functional tools

Exercises. Define the following functions.

1. A function `positive_places(f: Callable[[float], float], xs: Sequence[float]) -> list[float]` that takes as arguments some function `f` and a list of numbers `xs` and returns a list of those-and-only-those elements `x` of `xs` for which `f(x)` is strictly greater than zero.

   Example 1:

   ```
   def my_f(x: float) -> float:
       return x ** 3

   >>> positive_places(my_f, [1, 2, -1, -2, 3, 42, -9])
   [1, 2, 3, 42]

   >>> positive_places(my_f, [1, 2, 3, 4, 5])
   [1, 2, 3, 4, 5]

   >>> positive_places(my_f, [])
   []

   >>> positive_places(my_f, [-1, -2, -3, -4, -5])
   []
   ```

   Example 2:

   ```
   def f(x: float) -> float:
       return 2 * x + 4

   >>> positive_places(f, [10, 1, -3, -1.5, 0, 0.5])
   [10, 1, -1.5, 0, 0.5]
   ```

2. Write a function `eval_f_0123[A](f: Callable[[int], A]) -> list[A]` that evaluates the function `f` at positions `x=0`, `x=1`, `x=2` and `x=3`. The function should return the list `[f(0), f(1), f(2), f(3)]`.

   Example 1:

   ```
   def square(x: int) -> int:
       return x * x

   >>> eval_f_0123(square)
   ```

1. *Python Programming Primer*, Hans Fangohr *et al.* University of Southampton (2016)

```
[0, 1, 4, 9]
```

Example 2:

```
def cubic(x: int) -> int:
    return x ** 3

>>> eval_f_0123(cubic)
[0, 1, 8, 27]
```

Example 3:

```
def stars(x: int) -> str:
    return '*' * x

>>> eval_f_0123(stars)
['', '*', '**', '***']
```

3. A function `map[A, B](f: Callable[[A], B], xs: Sequence[A]) -> list[B]` which takes a function `f` and a list `xs` of values that should be used as arguments for `f`. The function `map` should apply the function `f` subsequently to every value `x` in `xs`, and return a list `fs` of function values, i.e. for an input argument `xs = [x0, x1, x2, ..., xn]` the function `map(f, xs)` should return `[f(x0), f(x1), f(x2), ..., f(xn)]`.

   Example 1:

```
def square(x: int) -> int:
    return x * x

>>> map(square, [-1, 10, 20, 42])
[1, 100, 400, 1764]
```

   Example 2:

```
>>> import math

>>> map(math.sqrt, [1, 2, 4, 9])
[1.0, 1.4142135623730951, 2.0, 3.0]
```

   Example 3:

```
def sign(x: float) -> int:
    if x > 0:
        return 1
    elif x < 0:
        return -1
    else:
        return 0

>>> sign(-1.1)
-1

>>> sign(0.1)
1

>>> sign(0.0)
0

>>> map(sign, [-0.2, -0.1, 0, 0.1, 0.2, 0.3])
[-1, -1, 0, 1, 1, 1]
```

4. A function `sum_f(f: Callable[[int], int], xs: Sequence[int]) -> int` that returns the sum of the function values of `f` evaluated at values `x0, x1, x2, ..., xn` where `xs = [x0, x1, x2, ..., xn]`.

   Example 1:

   ```
   def f(x: int) -> int:
       return x

   >>> sum_f(f, [1, 2, 3, 10])
   16
   ```

   Example 2:

   ```
   def square(x: int) -> int:
       return x * x

   >>> sum_f(square, [1, 2, 3, 10])
   114
   ```