[CS 11] Prac 7a – Squares

Submit [CS 11] Practice 7

My submissions

Points: 10 (partial)
Time limit: 6.0s
Memory limit: 1G

Author: kvatienza (Kevin Atienza)

Problem type

Allowed languages
NONE, py3

Problem Statement

Given a sequence of integers, give their squares.

Task Details

Your task is to implement a function called `squares`. This function has a single parameter: an iterable of `int`s.

The function must return a *generator* that generates `int`s. The `int`s must be the squares of the elements of the input sequence.

👌👌 **READ ME!!** 👌👌

Generator-Based Problems

This problem is meant solely for testing, and it is meant for you to learn how to solve generator-based problems in OJ.

In Python, an **iterable** is any value that can be "iterated over", e.g., with a `for` loop or a comprehension. Generators and sequences are examples of iterables.

Note that your generator must be **as lazy as possible**. It should yield each resulting next element as soon as it has enough information, and it should produce these results while advancing the input generators for as little as possible.

Do not take the next element unless necessary to yield the next result! In particular, it is (usually) a mistake to consume the input generators and collect them into concrete lists as a first step (unless you really have to). It is also a mistake to "look ahead" of the generators unnecessarily, if that isn't needed to yield the next result.

Also, note that the sequence(s) passed are sometimes generators, and sometimes non-generator sequence types. Non-generators cannot be passed to the `next` function.

Tasks

For this item, kindly do the following:

1. Submit each of the **correct** solutions below and verify that the verdict is `AC`.
2. Submit each of the **incorrect** solutions below and verify that the verdict is not `AC`, and the verdict is what you'd expect.
3. Be sure to understand each solution below!

Correct:

```
def squares(input_seq):
    for v in input_seq:
        yield v**2
```

Correct:

```
def squares(input_seq):
    return (v**2 for v in input_seq)
```

Correct:

```
def squares(input_seq):
    yield from (v**2 for v in input_seq)
```

Correct:

```
def squares(input_seq):
    input_seq = iter(input_seq)
    while True:
        try:
            yield next(input_seq)**2
        except StopIteration:
            break
```

Incorrect:

```
def squares(input_seq):
    # wrong! not lazy
    return [v**2 for v in input_seq]
```

Incorrect:

```
def squares(input_seq):
    # wrong! `list` fully consumes the input sequence
    for v in list(input_seq):
        yield v**2
```

Incorrect:

```
def squares(input_seq):
    # wrong! fails if input_seq is not a generator
    while True:
        try:
            yield next(input_seq)**2
        except StopIteration:
            break
```

Incorrect:

```
def squares(input_seq):
    # wrong! Looks ahead unnecessarily. It processes 5 elements at a time
    input_seq = iter(input_seq)
    while True:
        next_few = []
        for _ in range(5):
            try:
                next_few.append(next(input_seq))
            except StopIteration:
                break

        yield from (v**2 for v in next_few)

        if not next_few:
            break
```

Restrictions

In this lab session, many names are banned. Here are a few of them: `input`, `type`. This is *not* an exhaustive list. (If you accidentally use a variable name that turns out to be banned, please rename it.)

The following names are now allowed: `reversed`, `zip`.

The following import is now allowed: `count` from `itertools`.

`while` loops, `break`, and `continue` are now allowed.

This lab session focuses on generators. Therefore, recursion is banned.

For this problem:

- Loops and lists are allowed.
- Up to 8 function definitions are allowed.
- Recursion is **disallowed**. (The recursion limit has been greatly reduced.)
- Sets and dictionaries are allowed.
- Generators and comprehensions are allowed.
- The source code limit is 500.

Example Calls

NOTE:

In future generator-based problems, various example calls like these will not be given.

Example 1 Function Call

```
[*squares((1, 1, 4))]
```

Example 1 Return Value

```
[9, 1, 16]
```

Example 1 Explanation

Note that `squares` must be a generator; it was only converted to a `list` here for testing purposes.

Example 2 Function Call

```
print(*squares(iter((3, 1, 4))))
```

Example 2 Output

```
9 1 16
```

Example 3 Function Call

```
next(squares(iter((3, 1, 4))))
```

Example 3 Return Value

```
9
```

Constraints

- The function `squares` will be called at most 200 times.
- At most 500 elements will be consumed from the returned generator.
- Each element of the input sequence is a positive integer at most 10^{10}.

Scoring

- You get 100❤️ points if you solve all test cases.

Clarifications

Report an issue

No clarifications have been made at this time.