W10.2 Applied

Printers and generators

This week you have encountered generators as a way to produce iterators. This activity shows that generating a sequence is very similar to just printing a sequence.

Task 1

Write a function `print_divisible_by_three(lst)` that takes a list `lst` of integers and prints those that are divisible by 3.

For example, calling `print_divisible_by_three([2, 3, 5, 12, 8, 9, 4, 6])` should print

```
3
12
9
6
```

✓ Press the **Mark** button to see if you got it right.

Task 2

Modify your function `print_divisible_by_three()` so that instead of a list it works with an arbitrary *iterable* that produces integers.

For example, calling `print_divisible_by_three(range(7, 17))` should print

```
9
12
15
```

Did you have to modify your code at all to make this work? If you did, will your modified code still pass the test for Task 1?

✓ Press the **Mark** button to find out.

Task 3

Write a *generator* `generate_divisible_by_three(it)` that receives an iterator `it` producing integers and yields all those that are divisible by 3.

You should be able to use that generator like this ...

```
for n in generate_divisible_by_three(range(7, 17)):
    print(n)
```

... to produce the output

```
9
12
15
```

Hint: You should be able to use your code for Task 2 and modify only one line (and the name of the function, of course).

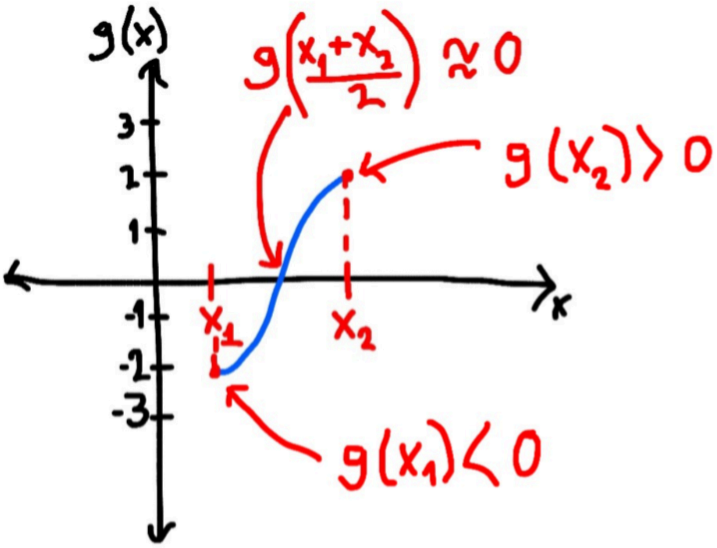
You should now be able to pass all the tests when you press **Mark**.

Polynomial Evaluator Revisited

In Week 10 Workshop, you have written a program that approximates the zeros of function g over an interval by only generating values of x that make $|g(x)| \leq \epsilon$ (i.e., let's say for the small value of epsilon $\epsilon = 0.0001$). This approach required us to arbitrarily choose the value of ϵ beforehand. Now, we will revisit the same task without needing to choose the value of ϵ.

Question 1

Write a function named `interval_x_zeros` that approximates the zeros of g over an interval. That is, `interval_x_zeros` function should take in the values of `init_x`, `delta_x`, `stop_x` and a function object `g` (e.g., a function returned by the previously defined `polynomial` function) as inputs (in that order) and produce a `generator` object that only generates values of x that make $g(x) \approx 0$. Specifically, we will assume $g(\frac{x_1 + x_2}{2}) \approx 0$ for two values of x that are very close to each other, say x_1 and $x_2 = x_1 + \Delta_x$, if $g(x_1) \times g(x_2) \leq 0$.



Intuition: For two values of x that are very close to each other, say x_1 and $x_2 = x_1 + \Delta_x$, if $g(x_1) \times g(x_2) \leq 0$ we will assume $g(\frac{x_1 + x_2}{2}) \approx 0$ because g is continuous. The reason is simple; let us first note that if $g(x_1)$ and $g(x_2)$ are on the opposite sides of the x-axis (e.g., as visualised above where $g(x_1)$ is negative and $g(x_2)$ is positive) their multiplication would be nonpositive. Moreover, we can also visually verify from inspecting the blue line visualised above that g must cross the x-axis (i.e., $g(\frac{x_1 + x_2}{2}) \approx 0$) somewhere between x_1 and x_2 (e.g., $\frac{x_1 + x_2}{2}$) to connect $g(x_1)$ and $g(x_2)$.

▶ Expand

Press **Mark** to check your implementation for Question 1.

Question 2

Write a class named `Interval_X_zeros` that is an iterator solving the tasks specified in Question 1. The class `Interval_X_zeros` should have the following four instance variables:

- `delta_x` denoting the value with which x will be incremented.
- `stop_x` denoting value until which x will be incremented.
- `x` denoting the current value of x.
- `g` denoting the polynomial g.

The constructor should take in the values of `init_x`, `delta_x`, `stop_x` and `g` as inputs (in that order) and store those values as instance variables (i.e., `self.delta_x = delta_x`, `self.stop_x = stop_x` and `self.g = g`) and also set the value of `init_x` to `x` (i.e., `self.x = init_x`). Moreover, the class should implement both `__iter__` and `__next__` methods.

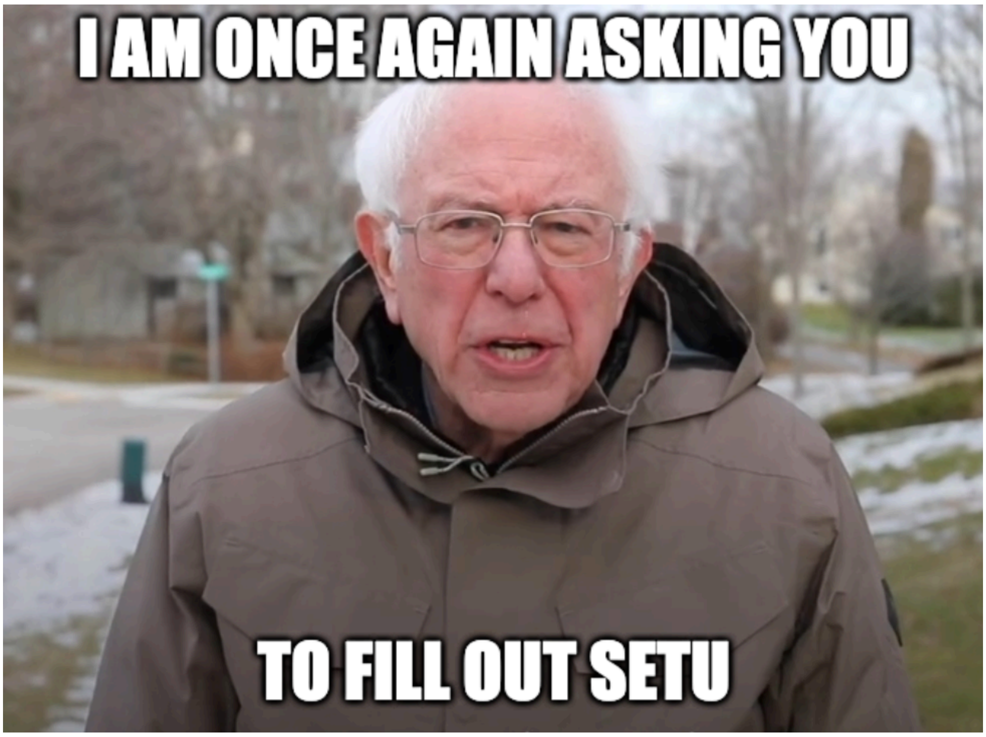
▶ Expand

✓ Press **Mark** to check your implementation for Question 2.

If you have finished early...

... please use this time to go back to the previous weeks' applied content you have not finished yet, and get that green tick!

Finally, please make sure to fill out SETU here.


I AM ONCE AGAIN ASKING YOU
TO FILL OUT SETU

Feedback

Question 1

What worked best in this lesson?

No response

Question 2

What needs improvement most?

No response