

Exercise: Iterators and Generators

Problems for exercise and homework for the [Python OOP Course @SoftUni](#).

Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/1945>.

1. Take Skip

Create a **class** called **take_skip**. Upon initialization, it should receive a **step** (int) and a **count** (int). Implement the **__iter__** and **__next__** functions. The iterator should return the **count** numbers (**starting from 0**) with the **given step**. For more clarification, see the examples:

Note: Submit only the class in the judge system

Examples

Test Code	Output
<pre>numbers = take_skip(2, 6) for number in numbers: print(number)</pre>	0 2 4 6 8 10
<pre>numbers = take_skip(10, 5) for number in numbers: print(number)</pre>	0 10 20 30 40

2. Dictionary Iterator

Create a class called **dictionary_iter**. Upon initialization, it should receive a **dictionary** object. Implement the iterator to return **each key-value pair** of the dictionary as a **tuple of two elements** (the key and the value).

Note: Submit only the class in the judge system

Examples

Test Code	Output
<pre>result = dictionary_iter({1: "1", 2: "2"}) for x in result: print(x)</pre>	(1, '1') (2, '2')
<pre>result = dictionary_iter({"name": "Peter", "age": 24}) for x in result: print(x)</pre>	("name", "Peter") ("age", 24)

3. Countdown Iterator

Create a class called **countdown_iterator**. Upon initialization, it should receive a **count**. Implement the **iterator** to return **each countdown number** (from **count** to **0** inclusive), separated by a single space.

Note: Submit only the class in the judge system

Examples

Test Code	Output
<pre>iterator = countdown_iterator(10) for item in iterator: print(item, end=" ")</pre>	10 9 8 7 6 5 4 3 2 1 0
<pre>iterator = countdown_iterator(0) for item in iterator: print(item, end=" ")</pre>	0

4. Sequence Repeat

Create a class called **sequence_repeat** which should receive a **sequence** and a **number** upon initialization. Implement an **iterator** to return the given elements, so they form a string with a length - the given **number**. If the **number is greater** than the number of elements, then the **sequence repeats** as necessary. For more clarification, see the examples:

Examples

Test Code	Output
<pre>result = sequence_repeat('abc', 5) for item in result: print(item, end='')</pre>	abcbab
<pre>result = sequence_repeat('I Love Python', 3) for item in result: print(item, end='')</pre>	I L

5. Take Halves

You are given a skeleton with the following code:

```
def solution():
    def integers():
        # TODO: Implement

    def halves():

        for i in integers():
            # TODO: Implement

    def take(n, seq):
        # TODO: Implement

    return (take, halves, integers)
```

Implement the **three** generator functions:

- **integers()** - generates an **infinite** amount of **integers** (starting from **1**)
- **halves()** - generates the halves of those integers (each integer / 2)
- **take(n, seq)** - takes the **first n** halves of those integers

Note: Complete the functionality in the skeleton and submit it to the judge system

Examples

Test Code	Output
<pre>take = solution()[0] halves = solution()[1] print(take(5, halves()))</pre>	[0.5, 1.0, 1.5, 2.0, 2.5]
<pre>take = solution()[0] halves = solution()[1] print(take(0, halves()))</pre>	[]

6. Fibonacci Generator

Create a generator function called **fibonacci()** that generates the **Fibonacci numbers** infinitely. **The first two numbers** in the sequence are **always 0 and 1**. Each following Fibonacci number is created by the **sum** of the **current** number **with the previous one**.

Note: Submit only the function in the judge system

Examples

Test Code	Output
<pre>generator = fibonacci() for i in range(5): print(next(generator))</pre>	0 1 1 2 3
<pre>generator = fibonacci() for i in range(1): print(next(generator))</pre>	0

7. Reader

Create a generator function called **read_next()** which should receive a **different number** of arguments (all iterable). On each iteration, the function should return each element from each sequence.

Note: Submit only the function in the judge system

Examples

Test Code	Output
<pre>for item in read_next("string", (2,), {"d": 1, "i": 2, "c": 3, "t": 4}): print(item, end='')</pre>	string2dict

```
for i in read_next("Need", (2, 3), ["words", "."]):
    print(i)
```

N
e
e
d
2
3
words
.

8. Prime Numbers

Create a generator function called **get_primes()** which should receive a **list of integer numbers** and return a list containing only the **prime numbers** from the initial list. You can learn more about prime numbers from [here](#):

Note: Submit only the function in the judge system

Examples

Test Code	Output
<code>print(list(get_primes([2, 4, 3, 5, 6, 9, 1, 0])))</code>	<code>[2, 3, 5]</code>
<code>print(list(get_primes([-2, 0, 0, 1, 1, 0])))</code>	<code>[]</code>

9. Possible permutations

Create a generator function called **possible_permutations()** which should receive a **list** and return lists with all possible permutations between its elements.

Note: Submit only the function in the judge system

Examples

Test Code	Output
<code>[print(n) for n in possible_permutations([1, 2, 3])]</code>	<code>[1, 2, 3]</code> <code>[1, 3, 2]</code> <code>[2, 1, 3]</code> <code>[2, 3, 1]</code> <code>[3, 1, 2]</code> <code>[3, 2, 1]</code>
<code>[print(n) for n in possible_permutations([1])]</code>	<code>[1]</code>