# Exercise: Iterators and Generators

Problems for exercise and homework for the [Python OOP Course @SoftUni](https://softuni.bg/courses/python-oop). Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/1945>

## Take Skip

Create a **class** called take\_skip. Upon initialization it should receive a **step** (number) and a **count** (number). Implement the \_\_iter\_\_ and \_\_next\_\_ functions. The iterator should return the **count amount** of numbers (**starting** **from 0**) and with the **given step**. For more clarification, see the examples:

***Note: Submit only the class in the judge system***

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| numbers = take\_skip(2, 6)  for number in numbers:  print(number) | 0  2  4  6  8  10 |
| numbers = take\_skip(10, 5)  for number in numbers:  print(number) | 0  10  20  30  40 |

## Dictionary Iterator

Create a class called dictionary\_iter. Upon initialization it should receive a **dictionary** object. Implement the iterator, so it returns **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** |
| result = dictionary\_iter({1: "1", 2: "2"})  for x in result:  print(x) | (1, '1')  (2, '2') |

## Countdown Iterator

Create a class called countdown\_iterator. Upon initialization it should receive a **count**. Implement the **iterator**, so it returns **each number of the countdown** (from count to **0** inclusive).

***Note: Submit only the class in the judge system***

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| iterator = countdown\_iterator(10)  for item in iterator:  print(item, end=" ") | 10 9 8 7 6 5 4 3 2 1 0 |

## 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 in the judge system***

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| take = solution()[0]  halves = solution()[1]  print(take(5, halves())) | [0.5, 1.0, 1.5, 2.0, 2.5] |

## Fibonacci Generator

Create a generator function called fibonacci() that generates the **Fibonacci numbers** infinitely (**starting from 0**). Each Fibonacci number is created by the **sum** of the **current** number **with the previous.**

***Note: Submit only the function in the judge system***

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| generator = fibonacci()  for i in range(5):  print(next(generator)) | 0  1  1  2  3 |