Функции с переменным количеством аргументов

*args

```
def add(*numbers):
    return sum(numbers)
```

```
add(3, 4, 5, 6)  # теперь можем передавать любое количество аргументов!
```

```
add()  # от нуля до бесконечности
```

```
def add_and_multiply(*numbers, coeff):  # любые аргументы после автоматически с
    return sum(numbers) * coeff
```

```
add_and_multiply(1, 2, 3, 4, coeff=3)
```

**kwargs

```
def func(*args, **kwargs):
    print(args)  # это список
    print(kwargs)  # а это словарь!
```

```
func(1, 2, 3)
```

```
func(1, 2, a=1, b=3, c=45)
```

Изменяемые объекты как аргументы функций: что выведет код?

```
def add_to_list(el, dest):
    dest.append(el)
    # функция ничего не возвращает!
```

```
a = [1, 2, 3]
add_to_list(4, a)
print(a)
```

Обратите внимание: значение аргумента по умолчанию определяется один раз - в момент объявления функции (а не каждый раз, когда она запускается!), поэтому, если он изменяемый...

```
def add_to_list(el, list_of_els=[]): # так делать не надо!
    list_of_els.append(el)
    return list_of_els


a = add_to_list(1)
b = add_to_list(2)
c = add_to_list(3, [])
d = add_to_list(4, [5, 6])
print(a, b, c, d)


def add_to_list_2(el, list_of_els=None): # перепишем
    if list_of_els is None:
        list_of_els = []
    list_of_els.append(el)
    return list_of_els


a = add_to_list_2(1)
b = add_to_list_2(2)
c = add_to_list_2(3, [])
d = add_to_list_2(4, [5, 6])
print(a, b, c, d)
```

Хотелось бы уметь копировать объекты!

```
from copy import copy, deepcopy


def add_to_list_2(el, dest):
    dest1 = copy(dest)
    dest1.append(el)


a = [1, 2, 3]
add_to_list_2(4, a)
print(a)
```

copy - это поверхностное копирование: создаётся новый изменяемый объект, в который переносятся все части старого объекта

Поэтому если внутри старого объекта были другие изменяемые объекты (например, это был список списков), то они не скопируются, а перенесутся

```
a = [[1, 2], [3, 4]]
b = copy(a)
# того же самого можно достичь с помощью:
# b = a.copy()
# b = a[:]
```

```
# b = list(a)
b[0].append(3)
print(a)
```

deepcopy - это глубокое копирование: все части копируются

```
a = [[1, 2], [3, 4]]
b = deepcopy(a)
b[0].append(3)
print(a)
```

Проверка: не запуская ячейку, определите, что выведет код:

```
a = [1, [2, 3]]
b = a[:]
a[1].append(2)
b[1].append(3)
print(a, b)
```

Рекурсия:

```
def countdown(num=10):
    while num >= 0:
        print(num)
        num -= 1


countdown()
```

```
def countdown_recursion(num=10):
    print(num)
    if num > 0:
        countdown_recursion(num - 1)


countdown_recursion()
```

```
def endless_recursion(num=0):
    print(num, end="\r") # обратите внимание, так каждая следующая строчка
    # будет писаться поверх предыдущей
    endless_recursion(num + 1)


endless_recursion()  # RecursionError (если повезёт, т.к. память может закончиться раньше)
```

```
def countdown_recursion_safer(num=10):
    if num < 0:
        raise ValueError("Can't count down from a negative number")
    print(num)
    if num > 0:
        countdown_recursion(num - 1)
```

Задание для выполнения в классе: напишите функцию, которая возвращает n-е число Фибоначчи: 1) с помощью рекурсии 2) без помощи рекурсии

Числа Фибоначчи:
0, 1, 1, 2, 3, 5, ...
F(0) = 0, F(1) = 1,
F(n) = F(n - 1) + F(n - 2) (n > 1)

Функция main():

Часто основной код программы стоит писать в отдельной функции, которую обычно называют main(). Это позволяет писать его в любом месте программы. Необходимо только не забыть вызвать функцию main() в конце кода.

```
def main():
    f1()
    f2()

def f1():
    print("This is f1")

def f2():
    print("This is f2")

main()
```

Модули. Импортирование модулей.

```
!wget https://pkholyavin.github.io/mastersprogramming/print_nums.py
!wget https://pkholyavin.github.io/mastersprogramming/print_nums2.py
!wget https://pkholyavin.github.io/mastersprogramming/print_nums3.py
```

```
import print_nums
print_nums.f1()
print_nums.f2()
```

Импортируем конкретные вещи:

```
from print_nums import f1, f2
f1()
f2()
```

Импортируем всё подряд (так лучше не делать, потому что легко запутаться)

```
from print_nums import *
f1()
f2()
```

Импортируем с другим именем:

```
import print_nums as pn
pn.f1()
```

Получить названия всего, что лежит в импортируемом модуле:

```
import print_nums

dir(print_nums)
```

Весь код, который находится в импортируемом файле, исполняется!

```
import print_nums2
```

Но код, оформленный так, при импортировании запускаться не будет!

```
def main():
    print("This is main")

if __name__ == "__main__":
    main()
```

```
import print_nums3
print_nums3.f1()
```

Порядок, в котором Python ищет импортируемые модули:

1. Встроенные модули

2. Текущая папка
3. Директории, содержащиеся в системной переменной PYTHONPATH
4. Установленные модули

```
import os
print(os.getcwd()) # текущая директория
```

Чтобы установить модуль через командную строку, используем менеджер пакетов `pip`

```
!pip install parselmouth
```

```
!python -m pip install parselmouth
```

```
python -m pip install [options] <requirement specifier> [package-index-options] ... python -m pip install [options] -r <requirements file> [package-index-options] ... python -m pip install [options] [-e] <vcs project url> ... python -m pip install [options] [-e] <local project path> ... python -m pip install [options] <archive url/path> ...
```

https://pip.pypa.io/en/stable/cli/pip_install/

Докстринги:
https://peps.python.org/pep-0257/

```
"""
This is a module that does...
"""

def f1() -> None:
    """
    This is a function that prints out the string "This is f1" and returns None
    """
    print("This is f1")


print(f1.__doc__)
```

Module-level dunders

```
__all__ = ["f1", "f2"]
__author__ = "John Smith"
__version__ = "1.0.1"
```

Обработка исключений

```
a = 1 / 0  # ZeroDivisionError
```

```
a = [1, 2]
a[5] = 0  # IndexError
```

```
try:
    a = 1 / 0
except:
    print("Oops")
    a = 0
```

```
try:
    a = 1 / 0
except Exception:  # общее название всех исключений
    print("Oops")
    a = 0
```

```
try:
    a = 1 / 0
except ZeroDivisionError:  # это исключение вызывается при делении на 0
    print("Oops")
    a = 0
```

```
try:
    a = 1 / 0
except IndexError:  # это исключение вызывается при обращении к неправильному индексу
    print("Oops")
    a = 0
```

```
b = [1, 2]
# b = [1, 2, 0]
try:
    a = 1 / b[2]
except (IndexError, ZeroDivisionError):  # обрабатываем несколько типов исключений сразу
    print("Oops")
    a = 0
```

```
b = [1, 2]
# b = [1, 2, 0]
try:
    a = 1 / b[2]
except IndexError:  # обрабатываем несколько типов исключений по очереди
    print("Oops: out of range")
```

```
    a = 0
except ZeroDivisionError:
    print("Oops: divide by zero")
    a = 0


try:
    a = 1 / 0
except ZeroDivisionError as e:  # сохраняем текст исключения
    print("The following exception has occured:", e)
    a = 0


c = 0
# c = 1
try:
    a = 1 / c
except ZeroDivisionError:
    print("Oops")
    a = 0
else:
    print("Success")
finally:
    print("Finishing up")
```

finally выполняется в любом случае; здесь стоит писать код, который должен обязательно выполниться (например, закрывает файл)

Особенности употребления:

1. try/except стоит применять, когда вероятность ошибки довольно низкая. В противном случае лучше использовать if/else
2. Внутри try стоит писать только код, который может вызвать ошибку, ничего лишнего
3. Внутри finally И try не стоит писать return

Вызов исключений:

```
def merge_strings(a, b):
    """Merges two strings: abc, def -> adbecf"""
    if not isinstance(a, str) or not isinstance(b, str):
        raise TypeError("a and b should be strings")
    if len(a) != len(b):
        raise ValueError("a and b should be equal in length")
    return "".join(i + j for i, j in zip(a, b))


merge_strings("abc", "def")
```

```
merge_strings("abc", "defg")  # ValueError

merge_strings("abc", 1)  # TypeError
```

Список возможных исключений:

https://docs.python.org/3/library/exceptions.html

Важные

IndexError - индекс за границами

TypeError - неверный тип

UnicodeError - ошибка при обработке символов Unicode

ValueError - верный тип, но неверное значение

ZeroDivisionError - деление на 0

FileNotFoundError - не существует требуемого файла (папки)

Исключения, которые возникают, если вы неправильно оформили код:

SyntaxError - синтаксическая ошибка

```
print(+)
```

IndentationError - неправильные отступы

```
if True:
print("1")
```

NameError - несуществующее имя переменной

```
variable = 1
print(varable)
```

assert (служит для дебаггинга программы)

Не стоит использовать это выражение в финальных версиях кода

```
def merge_strings(a, b):
    assert len(a) == len(b)
```

```
    return "".join(i + j for i, j in zip(a, b))

merge_strings("123", "4567")  # AssertionError
```

Проверка: не запуская ячейку, определите, что выведет код:

```
a = list(range(10))
b = [i / a[0] for i in a]
```

Проверка: не запуская ячейку, определите, что выведет код:

```
print("123" - "234")
```

Общая практика:

1. Написать функцию, которая принимает на вход целое число n и выводит на экран "ёлочку" из звёздочек высотой n.

```
#          *
#         ***
#        *****
#       *******
#      *********
#     ***********
#    *************
#   ***************
#  *****************
# *******************

def tree(n: int) -> None:
    ...
```

2. Напишите функцию, которая принимает на вход строку text, содержащую текст на русском языке, и возвращает словарь, в котором ключи - все возможные сочетания букв русского алфавита длиной 2, а значения - сколько раз они встретились в тексте.

```
def count_letter_bigrams(text: str) -> dict:
    ...
```

4. Дано:
   (1) Последовательность пар случайных целых чисел от 1 до 100 (отсортированных по

возрастанию)

(2) Последовательность случайных вещественных чисел от 1 до 100 (в случайном порядке)

Задание:

Для каждой пары чисел a, b из последовательности (1) определить, какие числа из последовательности (2) лежат между a и b. По очереди вывести на экран каждую пару и числа для неё в отсортированном порядке.

Чтобы получить последовательности, запустите ячейки ниже.

```
# сгенерируем первую последовательность
import random
seq1_len = random.randint(5, 8)
seq1_nums = sorted(random.sample(range(100), seq1_len * 2))
seq1 = [[a, b] for a, b in zip(seq1_nums[::2], seq1_nums[1::2])]
print(seq1)
# сгенерируем вторую последовательность
seq2_len = random.randint(50, 80)
seq2 = [round(random.random() * 100, 4) for _ in range(seq2_len)]
print(seq2)
```

```
# пример вывода:
# [1, 5]
# [1.2345, 2.3456, 3.4567]
# [8, 12]
# [8.0001, 9.0002, 10.0003, 11.0004]
```

Задание на дом (расширение задания 2):

Напишите функцию, которая принимает на вход строку text, содержащую текст на русском языке, **и целое число n**, а возвращает словарь, в котором ключи - сочетания букв русского алфавита **длиной n**, а значения - сколько раз они встретились в тексте.

Постарайтесь модифицировать код так, чтобы пробелы, знаки препинания и капитализация не учитывались, т.е. считайте, что в строке `"г. Санкт-Петербург"` есть трёхбуквенное сочетание `"гса"`

Оформите для функции docstring и постарайтесь учесть возможные ошибки

```
def count_letter_ngrams(text: str, n: int) -> dict:
    ...
```