Вебинар №7
Python, базовый курс

Dictionary (словарь)

```
# варианты создания

my_dict = dict()   # --> пустой словарь {}

my_dict2 = {}   # --> пустой словарь {}

ages = {

   "Dmitry": 26,

   "Alex": 34

}   # --> словарь с ключами "Dmitry" и "Alex"


del ages["Dmitry"]   # --> удалит ключ Dmitry
из словаря
```

- Хранилище данных типа ключ-значение. Близко к классической структуре данных HashTable.
- Например, может выступать как мини база данных. Размер и уровень вложенности не ограничен, хранить можно объекты любого типа.
- Ключи уникальны в пределах одного словаря.
- В качестве ключей могут выступать любые *неизменяемые (immutable)* встроенные типы.

Dictionary (словарь)

- Обратиться к словарю по ключу dict['key1']
- Поменять значение в словаре по ключу dict['key1'] = 't'
- Добавить пару ключ-значение dict['key2'] = 10
- Проверить наличие ключа в словаре с помощью in
- **dict.keys()** вернет список ключей словаря
- **dict.values()** вернет список значений словаря

} **dict.items()** вернет ключ+значение

- **dict.update('key1':10, 'key2': 't')** обновляет значения – удобен, когда нужно сделать несколько замен
- **dict.get('key1', 'key2', …)** – возвращает значения для конкретных ключей
- **Ключами могут быть только неизменяемые встроенные типы!**

Задача:

Создайте словарь, в котором ключами будут числа от 1 до 10, а значениями эти же числа, возведенные в 4 степень.

Вернемся к циклам

1. На вход даны два списка. Выведите каждый элемент первого списка, если его нет во втором списке. В списках могут быть повторяющиеся элементы.
2. Наполните множество уникальными квадратами введенных пользователем чисел. Наполнение прекращается в тот момент, когда пользователь вводит 0. В конце выведите полученное множество на экран
3. Допустим вы положили в некий финансовый инструмент x у.е под y процентов годовых. Посчитайте через сколько лет сумма удвоится. Выведите как будет меняться сумма на счете от года к году в списке и количество лет, которое это займет.

Вернемся к циклам

4. Игра угадай число. При каждом запуске компьютер загадывает случайное целое число в диапазоне от 1 до 100. Напишите программу -- диалог, которая будет давать подсказки до тех пор пока число не угадано. В тот момент, когда число угадано, игра завершается.
Пример. Пусть загадано число 34.
Программа просит ввести целое число.
Вводим 50
Программа говорит меньше
Вводим 25
Прграмма говорит больше
итд...
34
Игра завершается

Вернемся к циклам

5. Пройдитесь в цикле for по данному списку. Если на четном индексе стоит нечетное число, то превратите его в четное вверх.
Пример.

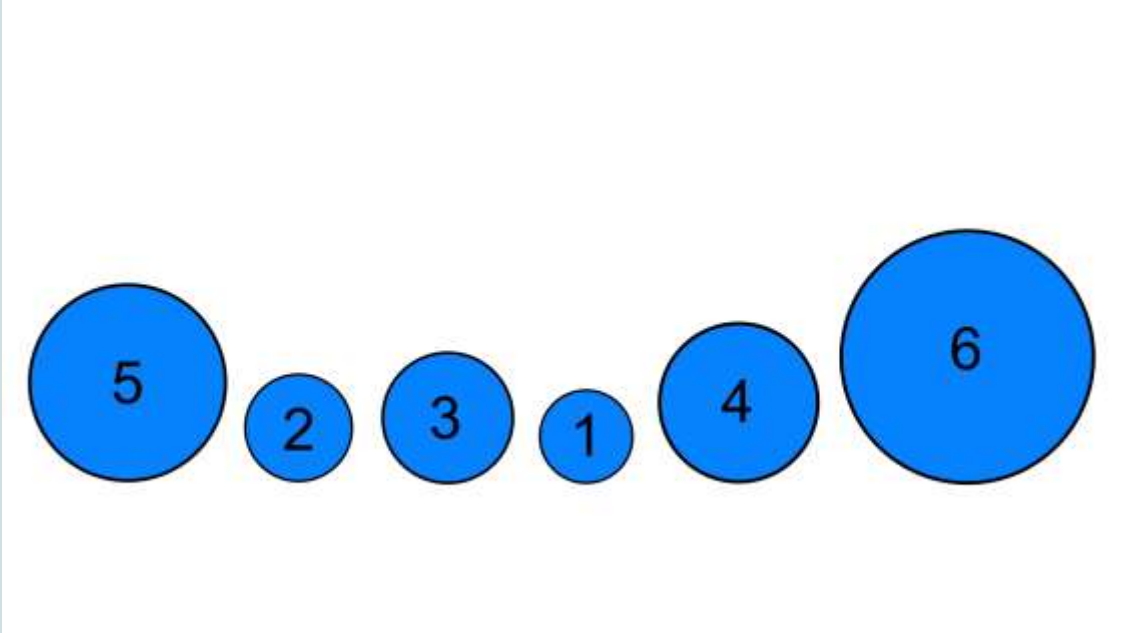
```
my_list = [1,2,3] --> [2, 2, 4]
```

6. Выведите лестницу высоты N из чисел. На каждом шаге ступенька должна состоять из чисел от 1 до i, где i - номер текущей итерации внешнего цикла.
7. Имеется 2 словаря с пересекающимися ключами (значения - положительные числа), напишите программу, которая формирует новый словарь по правилу:

Если в исходных словарях есть повторяющиеся ключи, выбираем среди их значений максимальное и присваиваем этому ключу (например, в словаре_1 есть ключ "a" со значением 5, и в словаре_2 есть ключ "a", но со значением 9. Выбираем максимальное значение, т. е. 9, и присваиваем ключу "a" в уже новом словаре)

Для самостоятельного кодинга

1. Простая реализация сортировки пузырьком. Реализуйте сортировку с помощью двух циклов for (один, вложен в другой) (например, по возрастанию).



Прерывание break

Оба цикла можно завершить досрочно. Для этого есть ключевое слово break

```
i = 0

while True:

    # делаем полезную работу

    if i > 2:

        break

    i += 1

for i in range(10):

    # делаем полезную работу

    if i > 5:

        break
```

Переход к следующему шагу continue

Можно досрочно завершить тело цикла и перейти к следующей итерации с помощью ключевого слова continue

```
i = 0

while True:

    i += 1

    if i == 1:

        continue  # пропускаем шаг, когда i будет равен 1


for i in range(10):

    if i == 5:

        continue  # пропускаем шаг, когда i будет равен 5
```

Задачи

1.Пусть пользователь бесконечно вводит числа до тех пор пока не введет 0. Проверяйте являются ли эти числа квадратами целых чисел, если нет, то возводите их в квадрат и добавляйте полученные квадраты в список, а если да, то сразу пишите их в список. В случае, если пользователь вводит числа больше 100, то добавлять в список не надо.

Пример.
5 →Не квадрат целого числа, возводим в квадрат и добавляем
16 →Добавляем
4 →Добавляем
900 → Больше 100, не добавляем
3 →Не квадрат целого числа, возводим в квадрат и добавляем
0
[25.0, 16.0, 4.0, 9.0]

Генератор списков и словарей

```
a = []
for i in range(1, 15):
    a.append(i)
print(a)
```

```
a = [i for i in range(1,15)]
print(a)
```

```
a = {}
for i in range(1, 15):
    a[i]=i**2
print(a)
```

```
a = {i: i**2 for i in range(1,15)}
print(a)
```

Функции

- это конструкции, которые:
  - получают на вход аргументы
  - выполняют некоторое действие или набор действий
  - [и возвращают результат]
- необходимы для того, чтобы при необходимости не писать одну и ту же логику несколько раз

$$k = 1 + \frac{r}{100}$$

$$limit = \frac{pay * (k^{term-1} - 1)}{(k-1) * k^{term}}$$

```
def annuity(pay,rate,term):
        k = 1+(rate/100)
        limit =(pay*(k**(term-1)-1))/((k-1)k**term)
        return limit
```

Функции. Синтаксис

Наименование функции, в нижнем регистре с разделителем _

Перечень аргументов, также могут быть указаны *args, **kwargs

Начинается с зарезервированного слова def

Тело функции

```
def annuity(pay,rate,term):
    k = 1+(rate/100)
    limit =(pay*(k**(term-1)-1))/((k-1)k**term)
    return limit
```

Зарезервированное слово для обозначения необходимости передачи результата функции во вне

Результат работы функции, который она возвращает

return:

- Если не указать, то функция ничего не вернет
- Если указать, но не уточнить что выводить, то вернет None

Задачи

1. Анаграммой строки S1 называется такая строка S2, которая получается из строки S1 путем перестановки символов. Напишите функцию, которая определяет является ли вторая строка анаграммой первой, возвращает YES, если является и NO, если не является.
2. Напишите функцию calc_sum(), которая на вход будет принимать целое положительное число, а возвращает сумму всех чисел от 0 до указанного аргумента, делящихся на 3 или на 5
3. Напишите функцию, которая на вход принимает 3 аргумента список, проверяемый элемент списка и номер вхождения и удаляет из списка искомый элемент на указанном вхождении. Если элемент входит в список меньше указанного числа раз, то функция должна возвращать исходный список.
4. Напишите функцию, которая на вход будет принимать два списка и возвращать один список, в котором будут чередоваться элементы исходных

Функции. Аргументы

```
def sum(a=20, b=30):
        c = a + b
        return c
```

- Значение по умолчанию:

```
sum() → 50
sum(a = 10) → 40
```

- Перестановка аргументов:

```
sum(b = 10, a = 20) → 30
sum(b = 10, 20) → Error
sum(10, a = 20) → Error
sum(10, 20) → 30
```

- *args – переменное число аргументов

```
def print(*args):
    return args
print(1, 2, 3) → (1, 2, 3)
```

- **kwargs – переменное число именованных аргументов

```
def print(*kwargs):
    return kwargs
print(a = 1, b = 2, c = 3) → {'a': 1, 'b':2, 'c':3)
```

Лямбда-функции

- анонимные функции, удобны для несложных алгоритмов
- синтаксис: `(lambda a, b: a+b)`
- использование:
  - разово `(lambda a, b: a+b)(0, 20)`
  - неоднократно

```
sum =(lambda a, b: a+b)

sum(10, 20)
```

Область видимости

Области видимости определяют, в какой части программы мы можем работать с той или иной переменной, а от каких переменная "скрыта".

Позволяют избежать конфликта имен при использовании наработок разных программистов в какой-то другой программе.

Если бы все имена были глобальными, то нельзя было бы безопасно соединить две подпрограммы, у которых определена переменная с одинаковым названием.

Программа 1: a = 3 ---

--------> Программа 3: Какое значение a нужно?

Программа 2: a = 10 ---

Правило LEGB

Правило разрешения имен в python

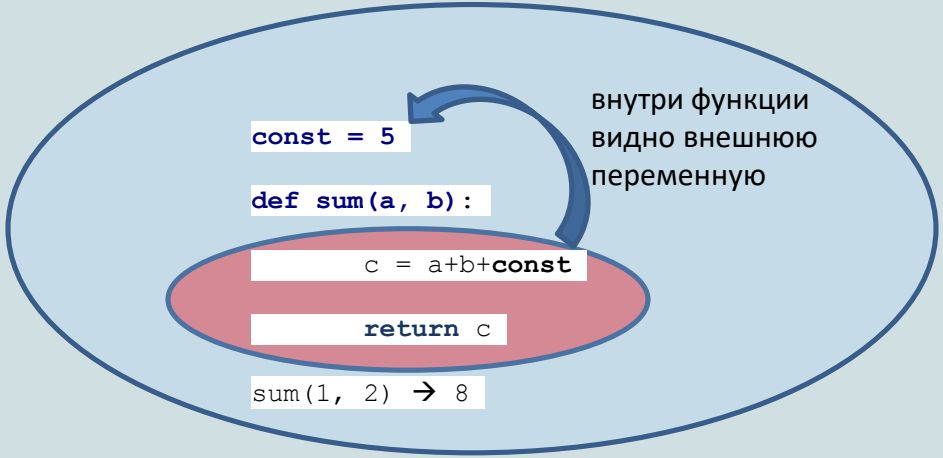
- **Область видимости local (function)**
- Область видимости enclosing (nonlocal)
- **Область видимости global**
- **Область видимости built-in**

Описывает стандартный механизм поиска "названий" во время выполнения программы.

Global (module) scope

scope = область видимости

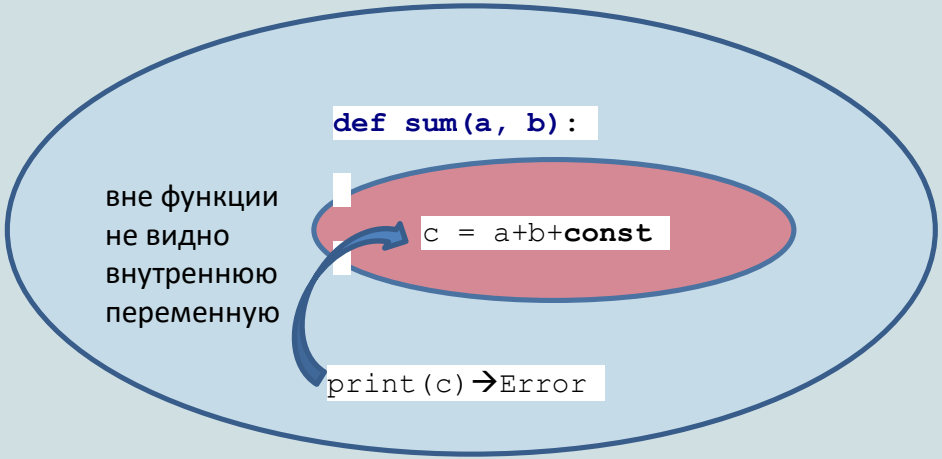
Область видимости модуля. Переменные вне функций, классов этого модуля. Например, когда мы просто пишем клетки кода в colab, чаще всего мы создаем имена в глобальной области видимости (если только не оформляем код в функции, а имена функций в свою очередь появляются в глобальном пространстве имен модуля)



Local (function) scope

scope = область видимости

Все имена переменных, которые мы определяем в функции. Они будут видны внутри блока кода функции, но не будут видны за ее пределами.



Пример 1

```
def sample_function(var_y):
    var_x = var_y**2
    print(var_y)
    return var_x

var_c = 10
sample_function(20)
```

Какие переменные здесь глобальные, а какие локальные?

Пример 2

```
def sample_function(var_a, var_b):
    print("В функции")
    print(f"var_a: {var_a}")
    print(f"var_b: {var_b}")
    print(f"var_c: {var_c}")
    var_d = 888

var_a = 3
var_c = 1
print("Снаружи функции")
print(f"var_a: {var_a}")
print(f"var_c: {var_c}")
sample_function(33, 100)
print(f"var_d: {var_d}")
```

Что будет выведено на экран?

Пример 3

```
def sample_function(var_a, var_b):
    print("В функции")
    print(f"var_a: {var_a}")
    var_a["z"] = 777
    print(f"var_b: {var_b}")
    var_b = (3, 3, 3)
    print(f"var_b: {var_b}")


var_a = {"x": 1, "y": 2}
var_b = (1,2,3)
print("Снаружи функции")
print(f"var_a: {var_a}")
print(f"var_b: {var_b}")
sample_function(var_a, var_b)
print("После функции")
print(f"var_a {var_a}")
print(f"var_b {var_b}")
```

Что будет напечатано на экран?

Built-in scope

Встроенная область видимости. Мы с ней уже сталкивались. Сюда входят все встроенные в язык функции, зарезервированные слова, исключения и т.д.

Например, int(), float(), dict(), def, list(), type(), …

Посмотреть имена, определенные в built-in можно распечатав dir(__builtins__)

Обратная связь

https://docs.google.com/forms/d/e/1FAIpQLScLLl5Vno4ynQVLn4Tk5CifvSxYueYxFXcWQfybxQ1UvPSoXQ/viewform?usp=sf_link