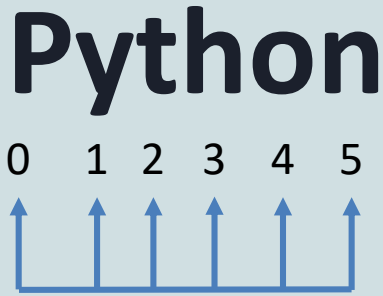
# Вебинар №6
# Python, базовый курс

# Методы строк в Python

1.Длина строки **len(string)** – возвращает число символов в строке: *len('Python') → 6*
2.Поиск подстроки **string.[r]find(str_find, [start], [end])** – возвращает индекс первого (**rfind** – последнего) вхождения: *'Python'.find('t')→2*
3.Замена подстроки **string.replace(str_find, str_replace):**
*'Python'.replace('t', 'ff')→'Phyffon'*
4.Приведение к верхнему/нижнему регистру **string.upper()/string.lower():**
*'Python'.upper() → 'PHYTHON'*
5.Проверка на различные условия **is%()** – возвращает True или False:
**string.isupper()/string.islower()/string.istitle()** – проверка что строка находится в верхнем регистре/нижнем/начинается с заглавной
**string.isdigit()/string.isalpha()/string.isalumn()** – проверка, что строка состоит из цифр/букв/цифр или букв
*'Python'.istitle()→ True*

# Индексы и строки




адрес каждой буквы = индекс

**Возвращать значение по индексу**

a = 'Python'
a[0]: 'P'
a[1]: 'y'
a[5]: 'n'
a[6]:

**Строки относятся к категории неизменяемых типов данных , поэтому**

a = 'Python'
a[0] = 's'

**А как тогда…?**

# Срезы – конкатенация "наоборот"

**Для решения таких задач используются срезы (слайсы)**

```
a = 'Python'
a[0:3]: 'Pyth'
a[1:]: 'ython'
a[:2]: 'Pyt'
a[1:-2]: 'yth'
a[2::3]: 'tn'
```

**string[first_index:last_index:[step]]**

**Возвращаемся к нашему примеру, хотим первый символ заменить на 's':**

```
a = 's'+a[1:]
```

# Задачи

1. Дана строка '2378O79213'
    1. Вывести с 5 по 7 символы
    2. Вывести длину строки
    3. Вывести предпоследний символ
    4. Проверить содержатся ли в строке буквы
    5. Вывести все символы с четными индексами
    6. Если в строке есть символ '3', то заменить его на '222'

2. Написать программу, которая принимает на вход 2 параметра – строку и подстроку и производит поиск подстроки в строке. Если находит, то выводит 'Бинго', иначе 'Увы'. Вхождение должно быть найдено независимо от регистра

# Строки и циклы

1.Строки состоят из подстрок
2.К каждой подстроке можем обратиться как к строке
3.Как это сделать "минимальными" усилиями?

**Циклы**

- for (без условия)
- while (с условием)

# for (делай n раз или пока коллекция не исчерпает себя)

```
for <переменная/е> in <что-то по чему можно итерироваться>:

. . . .<делаем что-то с переменной/ыми>
```

P.S.: range([start], end, [step]) – создает ряд чисел

**Пример 1**

```
for i in range(5):
    print(i)
```

**Пример 2**

```
a = 'Python'

for index in range(len(a)):
    print(index, a[index])
```

- Часто используется для прохода по какой-нибудь коллекции.
- Если мы проходимся по какому-то объекту, например, списку, то не следует изменять размер этого объекта в теле цикла.
- Использовать можно тот цикл, который вам удобнее, но чаще всего for выглядит лаконичнее

# while (делай пока условие истинно)

```
while <Условие истинно (True)>:
    <делай что-то>
```

Пример 1

```
i = 1
while i < 3:
    print(f"Hi {i}")
    i += 1
```

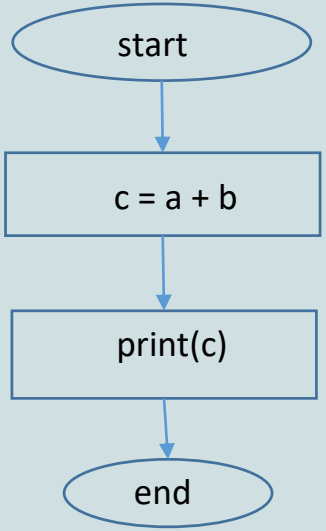
Пример 1

```
i = 0
A = 'Python'
Letter = ''
while letter != 't':
    letter = a[i]
    print(letter)
    i += 1
```

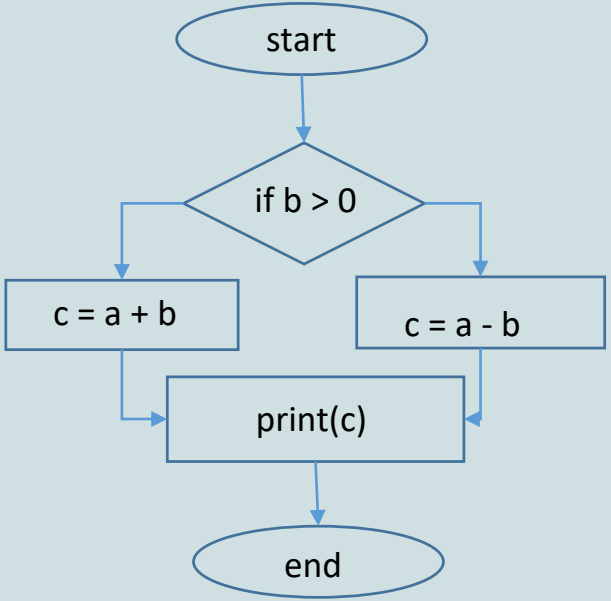
- Удобно в тех случаях, когда точно неизвестно сколько итераций нужно. Python выполняет тело цикла, пока какое-то условие не перестанет быть истинным.
- Иногда можно увидеть while True: т.е бесконечный цикл

# Алгоритмы

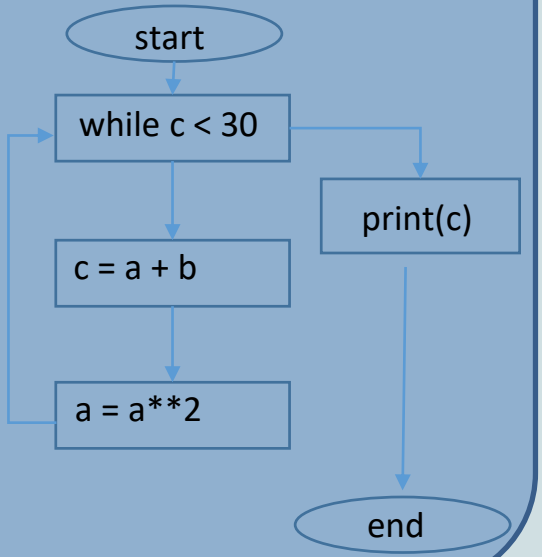
**Линейные**
команды выполняются последовательно




**С ветвлением**
алгоритм имеет несколько веток решения в зависимости от значений параметров




**Циклические**
команды повторяются до конца цикла



# Встроенные типы

- 'Простые'
  - Numbers (числа)
    - Целые числа (int)
    - Числа с плавающей точкой (float)
    - Комплексные числа
  - Strings (строки) (str)
  - Boolean (логический тип данных) (bool)

- **Коллекции**
  - **Lists (списки)**
  - **Dictionaries (словари)**
  - **Tuples (кортежи)**
  - **Sets (множества)**

# List (список)

```
# варианты создания

my_list = list() # --> пустой список

my_list = [] # --> пустой список

my_list = [1,2,3] # --> список [1,2,3]

my_list[0] # --> 1

my_list[:2] # --> [1, 2]
```

- Удобны для хранения упорядоченного множества объектов.
- Часто удобны для поэтапного наполнения.
- Динамические, неограниченные по размеру, могут хранить произвольные объекты.
- Есть встроенная целочисленная индексация и слайсинг
- Методы пополнения, вставки, сортировки …

# List (список)

**Методы строки и списка:**
**string.split('sep') → list с разделителем 'sep'**
**'sep'.join(list/itter) → string  с разделителем 'sep'**

**Методы списка:**

- **list_variable.append(element)** – добавляет элемент в конец списка
- **list_variable.extend(list2)** – добавляет список в конец исходного списка
- **list_variable.insert(index, element)** – вставляет в список элемент на указанный индекс
- **list_variable.remove(element)** – удаляет из списка первый найденный элемент
- **list_variable.pop(index)** – удаляет из списка элемент с указанным индексом и передает удаленное значение на экран. Если индекс не найден, то будет удален последний элемент
- **list_variable.index(element)** – возвращает номер индекса первого найденного элемента в списке
- **list_variable.sort([reverse=True])** – сортирует значения в списке
- **list_variable.count(element)** – кол-во вхождений определенного символа
- С помощью конструкции in можно проверить есть ли в списке искомый элемент

# Задачи

1. В данном списке l = [1, 5, 3, 100] найдите значение 5 и замените его на значение 55.
2. В данном списке, возьмите каждый третий элемент, начиная с элемента с индексом 4 и до индекса 15 (невключительно).
3. Переверните строку, поступающую на вход. Проверьте, что строка является палиндромом (прямой порядок букв равен обратному).
4. Примите на вход 3 пользовательских ввода (3 произвольных строки). Добавьте каждый ввод в список, если его еще нет в этом списке.

# Tuple (кортеж)

```
# варианты создания

my_tuple = tuple()  # --> пустой кортеж

my_tuple = ()  # --> пустой кортеж

my_tuple = (1, 2, 3, 4, 5)  # --> кортеж (1, 2, 3, 4, 5)

# индексация и слайсинг

my_tuple[1]  # --> 2

my_tuple[2:4]  # --> (3, 4)
```

- Похож на list, НО является неизменяемым: т.е нельзя ни удалить, ни добавить в него новые элементы
- Зачем же он такой нужен? Используются для:
  - защиты данных от пользователей
  - ускорения процесса считывания данных

# Tuple (кортеж)

- Всегда можем перейти от кортежа к списку и наоборот:
  - list_variable = list(tuple_variable)
  - tuple_variable = tuple(list_variable)

- Работают те же методы, что и для списков (за исключением изменяющих кортеж)

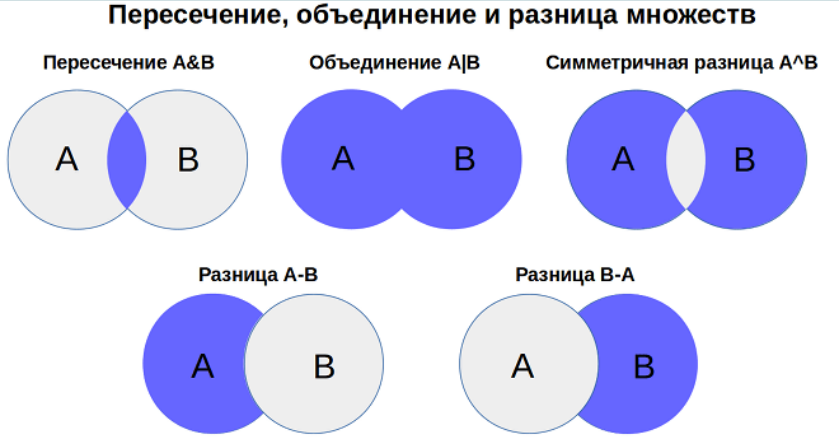
# Set (множество)

```
# варианты создания

my_set = set()   # --> пустое множество

my_set2 = {1, 2, 3}   # --> множество из
элементов {1, 2, 3}

my_set3 = {1, 2, 2, 3}   # --> множество из
элементов {1, 2, 3}

my_set3.add(4)   # --> добавляет 4 к множеству

my set3.pop()   # --> достает из множества
случайный элемент
```

- *Неупорядоченный* набор **уникальных** объектов.
- Удобно в ситуациях, когда нужно обработать некий поток данных из которого нужно взять только уникальные значения.
- Или когда нужно произвести операции выборки определенного подмножества из нескольких данных. Например, взять только те значения, которые есть в наборе a и наборе b
- Может хранить внутри себя неизменяемые типы

# Set (множество)

- Мощность множества **len(set)**
- Добавление элемента в множество **set.add(element)**
- Удаление элемента по значению **set.discard(element)**
- Удаление произвольного элемента и передает удаленное значение на экран **set.pop()**



# Задачи

1. Напишите программу, которая получает на вход список элементов или строку и преобразует их в множество

2. Дан простой текст. Определите сколько в тексте различных слов
P.S.: пробелы, точки и запятые не считаются за слова
P.S.: необходимо учесть, что слова бывают в разных регистрах

3. Класс отправился за грибами, кто-то нашел белые, кто-то мухоморы, а кто или и то, и другое. 25 учеников нашли белые, 9 мухоморы, 13 - и белые, и мухоморы. Сколько детей в классе?

# 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.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!**

# Задачи

1. Создайте словарь, в котором ключами будут числа от 1 до 10, а значениями эти же числа, возведенные в 4 степень.
2. Создайте словарь с количеством элементов не менее 5-ти.
   Поменяйте местами первый и последний элемент объекта.
   Удалите второй элемент.
   Добавьте в конец ключ «new_key» со значением «new_value».
   Выведите на печать итоговый словарь. Важно, чтобы словарь остался тем же (имел тот же адрес в памяти).

# Вернемся к циклам

1. Наполните множество уникальными квадратами введенных пользователем чисел. Наполнение прекращается в тот момент, когда пользователь вводит 0. В конце выведите полученное множество на экран
2. 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 год от году в списке и количество лет, которое это займет.
3. Пусть дан список из N чисел. Возведите каждое число в этом списке в квадрат и выведите на экран.
4. На вход даны два списка. Выведите элемент первого списка, если его нет во втором списке

# Вернемся к циклам

5. 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

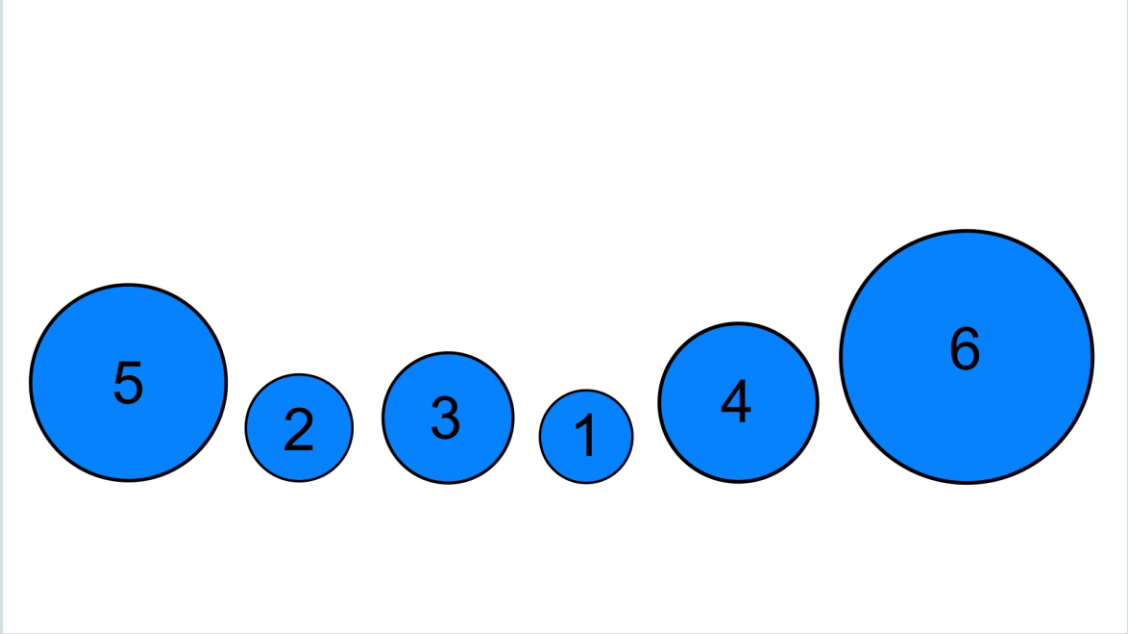
6. Пройдитесь в цикле for по данному списку. Если на четном индексе стоит нечетное число, то превратите его в четное вверх.
Пример.

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

7. 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.

# Для самостоятельного кодинга

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



# Обратная связь

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