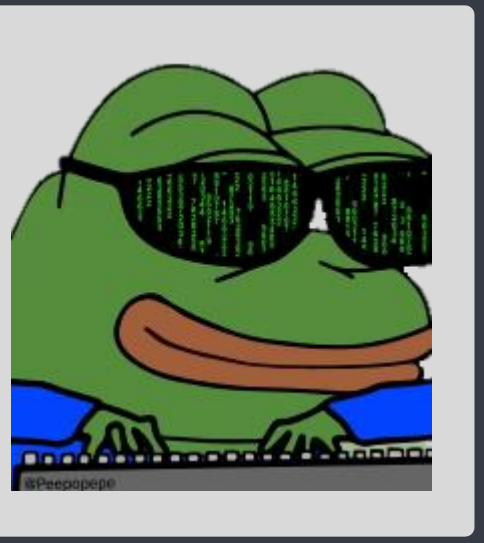
План на занятие:

1. VS Code text editor
2. Основные итерируемые объекты в Python и их возможности
   - 2.1. Список (list)
   - 2.2. Строка (str)
3. Итерации
   - 3.1. For .. in ..
   - 3.2. While ..
4. Логические выражения, алгебра логики (?)

План на занятие:

1. VS Code text editor
2. Основные итерируемые объекты в Python и их возможности
   - 2.1. Список (list)
   - 2.2. Строка (str)
3. Итерации
   - 3.1. For .. in ..
   - 3.2. While ..
4. Логические выражения, алгебра логики (?)

Code



Code

План на занятие:

1. VS Code text editor
2. Основные итерируемые объекты в Python и их возможности
   - 2.1. Список (list)
   - 2.2. Строка (str)
3. Итерации
   - 3.1. For .. in ..
   - 3.2. While ..
4. Логические выражения, алгебра логики (?)

Типы данных: список (list)

Позволяет хранить массив данных:

```
l = []                                  # пустой список
l = [1, 2, 3]                           # список чисел
l = ["Иван", 621, "экономика"]          # список различных типов данных
l = list("spam")                        # список элементов итерируемого объекта
```

Типы данных: список (list)

Позволяет хранить массив данных:

```
l = []                              # пустой список
l = [1, 2, 3]                       # список чисел
l = ["Иван", 621, "экономика"]      # список различных типов данных
l = list("spam")                    # список элементов итерируемого объекта
```

Основные возможности взаимодействия со списками:

- Доступ к произвольному элементу списка – индексация, срез
- Изменение элемента списка – присваивание по индексу
- Объединение списков – конкатенация, повторение
- Проход по всем элементам списка – итерация
- Добавление и удаление элементов – увеличение, уменьшение

Типы данных: список (list)

Позволяет хранить массив данных:

```
l = []                              # пустой список
l = [1, 2, 3]                       # список чисел
l = ["Иван", 621, "экономика"]      # список различных типов данных
l = list("spam")                    # список элементов итерируемого объекта
```

Основные возможности взаимодействия со списками:

- Доступ к произвольному элементу списка – индексация, срез
- Изменение элемента списка – присваивание по индексу
- Объединение списков – конкатенация, повторение
- Проход по всем элементам списка – итерация
- Добавление и удаление элементов – увеличение, уменьшение

Типы данных: список (list)

Индексация:

- получение произвольного элемента списка

Синтаксис:

```
list_object[index]
```

Типы данных: список (list)

Индексация:

- получение произвольного элемента списка

Синтаксис:

```
list_object[index]
```

Примеры:

1.
```
l = [15, 24, 3]
l[1]  # 24
```

Типы данных: список (list)

Индексация:

- получение произвольного элемента списка

Синтаксис:

```
list_object[index]
```

Примеры:

```
2. l = [
        ["J", "agent_J@mib.com", 1],
        ["K", "agent_K@mib.com", 0]
   ]
   l[1][0]  # "K"
```

Типы данных: список (list)

Индексация:

- получение произвольного элемента списка

Синтаксис:

```
list_object[index]
```

Примеры:

3.
```
l = [[1, 4, 7],
     [2, 5, 8],
     [3, 6, 9]]
i = 2
j = 0
l[i][j]  # 3
```

Типы данных: список (list)

Индексация:

- получение произвольного элемента списка

Синтаксис:

```
list_object[index]
```

Примеры:

4.
```
l = list("economics")
l  # ["e", "c", "o", "n", "o", "m", "i", "c", "s"]
l[-1].upper()  # "S"
l[len(l) - 1].upper()  # "S"
```

Типы данных: список (list)

Срез:

- получение среза списка

Синтаксис:

```
list_object[start:end]
```

-> interval [l[start], ..., l[end])

Типы данных: список (list)

Срез:

- получение среза списка

Синтаксис:

```
list_object[start:end]
```

-> interval [l[start], ..., l[end])

Длина среза равна end – start

тогда:

len(l[start:end]) == end – start

Типы данных: список (list)

Срез:

- получение среза списка

Синтаксис:

```
list_object[start:end]
```

-> interval [l[start], ..., l[end])

Примеры:

```
1. [15, 24, 3][:]  # [15, 24, 3]
```

Типы данных: список (list)

Срез:

- получение среза списка

Синтаксис:

```
list_object[start:end]
-> interval [ l[start], ..., l[end] )
```

Примеры:

2.
```
l = [
        ["J", "agent_J@mib.com", 1],
        ["K", "agent_K@mib.com", 0]
    ]
l[1][:2]  # ["K", "agent_K@mib.com"]
```

Типы данных: список (list)

Срез:

- получение среза списка

Синтаксис:

```
list_object[start:end]
-> interval [ l[start], ..., l[end] )
```

Примеры:

```
3. l = [[1, 4, 7],
        [2, 5, 8],
        [3, 6, 9]]
   l[:][1] == a[1]  # [2, 5, 8]
```

Типы данных: список (list)

Срез:

- получение среза списка

Синтаксис:

```
list_object[start:end]
-> interval [ l[start], ..., l[end] )
```

Примеры:

4.
```
l = list("economics")
l  # ["e", "c", "o", "n", "o", "m", "i", "c", "s"]
l[-5:] # ["o", "m", "i", "c", "s"]
l[-3:-1]  # ["i", "c"]
```

Типы данных: список (list)

Позволяет хранить массив данных:

```
l = []                              # пустой список
l = [1, 2, 3]                       # список чисел
l = ["Иван", 621, "экономика"]      # список различных типов данных
l = list("spam")                    # список элементов итерируемого объекта
```

Основные возможности взаимодействия со списками:

- Доступ к произвольному элементу списка – индексация, срез
- Изменение элемента списка – присваивание по индексу
- Объединение списков – конкатенация, повторение
- Проход по всем элементам списка – итерация
- Добавление и удаление элементов – увеличение, уменьшение

Типы данных: список (list)

Изменение элементов списка:

Синтаксис:

```
list_object[index] = object
list_object[start:end] = iterable_object
```

Типы данных: список (list)

Изменение элементов списка:

Синтаксис:

```
list_object[index] = object
list_object[start:end] = iterable_object
```

Примеры:

1.
```
l = [None, 'example@gmail.com', 'admin']
l[0] = 'big_boss'  # ['big_boss', 'example@gmail.com', 'admin']
```

Типы данных: список (list)

Изменение элементов списка:

Синтаксис:

```
list_object[index] = object
list_object[start:end] = iterable_object
```

Примеры:

2.
```
l = ['Sponge', 'Bob', 'Square pants']
l[:2] = ['Mr', 'Crabs']  # ['Mr', 'Crabs', 'Square pants']
```

Типы данных: список (list)

Позволяет хранить массив данных:

```
l = []                              # пустой список
l = [1, 2, 3]                       # список чисел
l = ["Иван", 621, "экономика"]      # список различных типов данных
l = list("spam")                    # список элементов итерируемого объекта
```

Основные возможности взаимодействия со списками:

- Доступ к произвольному элементу списка – индексация, срез
- Изменение элемента списка – присваивание по индексу
- Объединение списков – конкатенация, повторение
- Проход по всем элементам списка – итерация
- Добавление и удаление элементов – увеличение, уменьшение

Типы данных: список (list)

Конкатенация:

Синтаксис:

```
list_object + list_object
```

Типы данных: список (list)

Конкатенация:

Синтаксис:

```
list_object + list_object  # [ ] + [ ]
```

Пример:

```
1. a = [1, 2, 3]
   b = [-1, -2, -3]
   a + b  # [1, 2, 3, -1, -2, -3]
```

Типы данных: список (list)

Конкатенация:

Синтаксис:

```
list_object + list_object  # [ ] + [ ]
```

Пример:

```
2. list('hello') + list('world')  # ['h', 'e', 'l', 'l', 'o', 'w', 'o', 'r', 'l', 'd']
```

Типы данных: список (list)

Конкатенация:

Синтаксис:

```
list_object + list_object  # [ ] + [ ]
```

Пример:

```
films = [
    ['Inception', 2010],
    ['The prestige', 2006],
]
films += [['Tenet', 2020]]
films[-1]  # ['Tenet', 2020]
```

Типы данных: список (list)

Позволяет хранить массив данных:

```
l = []                              # пустой список
l = [1, 2, 3]                       # список чисел
l = ["Иван", 621, "экономика"]      # список различных типов данных
l = list("spam")                    # список элементов итерируемого объекта
```

Основные возможности взаимодействия со списками:

- Доступ к произвольному элементу списка – индексация, срез
- Изменение элемента списка – присваивание по индексу
- Объединение списков – конкатенация, повторение
- Проход по всем элементам списка – итерация
- Добавление и удаление элементов – увеличение, уменьшение

Типы данных: список (list)

Увеличение:

- добавление элементов в список

Синтаксис:

1. list_object.append(object)
2. list_object.extend(iterable) # == list_object + list(iterable)
3. list_object.insert(i, object)

Пример:

```
1. films = [
       ['Inception', 2010]
   ]
   films.append(['Tenet', 2020])  # [['Inception', 2010], ['Tenet', 2020]
```

Типы данных: список (list)

Увеличение:

- добавление элементов в список

Синтаксис:

1. list_object.append(object)
2. list_object.extend(iterable) # == list_object + list(iterable)
3. list_object.insert(i, object)

Пример:

2. [1, 2, 3].extend([4, 5, 6]) # [1, 2, 3, 4, 5, 6]

Типы данных: список (list)

Увеличение:

- добавление элементов в список

Синтаксис:

1. list_object.append(object)
2. list_object.extend(iterable) # == list_object + list(iterable)
3. list_object.insert(i, object)

Пример:

```
3. [
     [0, 3, 6],
     [2, 5, 8]
   ].insert(1, [1, 4, 7])[2]  # [2, 5, 8]
```

Типы данных: список (list)

Уменьшение:

- удаление элементов из списка

Синтаксис:

1. list_object.pop(index)
2. list_object.remove(object)
3. list_object[i:j] = []

Пример:

```
1. ratings = [13, 24, 72, 0]
   ratings.sort(reverse=True)
   top_rating = ratings.pop(0)  # == 72 == max(ratings)
```

Типы данных: список (list)

Уменьшение:

- удаление элементов из списка

Синтаксис:

1. list_object.pop(index)
2. list_object.remove(object)
3. list_object[i:j] = []

Пример:

```
2. film = [
        ['Inception', 2010],
        ['The prestige', 2006],
    ].pop(0)[0]  # 'Inception'
```

Типы данных: список (list)

Уменьшение:

- удаление элементов из списка

Синтаксис:

1. list_object.pop(index)
2. list_object.remove(object)
3. list_object[i:j] = []

Пример:

```
3. matrix = [[1, 2, 3], [2, 1, 2], [3, 2, 1]]
   matrix[:2] = []
   m[0]  # [3, 2, 1]
```

Типы данных: список (list)

Создание	[ ], list( )	
Индексация, срез	l[i], l[i:j]	
Конкатенация, повторение	l1 + l2, l*3	
Методы l.method(..)	Поиск	l.index(item)
	Увеличение	l.append(item), l.insert(i, item)
	Уменьшение	l.pop(i), l.remove(item)
	Сортировка	l.sort()
Изменяемость	l[i] = ...	

Типы данных: список (list)

План на занятие:

1. VS Code text editor
2. Основные итерируемые объекты в Python и их возможности
   - 2.1. Список (list)
   - 2.2. Строка (str)
3. Итерации
   - 3.1. For .. in ..
   - 3.2. While ..
4. Логические выражения, алгебра логики (?)

Типы данных: строка (string)

Позволяет хранить текстовые данные:

```
s = ''                  # пустая строка
s = 'name'              # строка из 4 символов
s = "name"              # та же строка из 4 символов
s = "('-')"             # кавычки создания строки могут хранить другие кавычки
```

Типы данных: строка (string)

Позволяет хранить текстовые данные:

```
s = ''                  # пустая строка
s = 'name'              # строка из 4 символов
s = "name"              # та же строка из 4 символов
s = "('-')"             # кавычки создания строки могут хранить другие кавычки

s = str(15)             # s == '15'
s = str([1,2,3])        # s == "[1, 2, 3]"
```

Типы данных: строка (string)

Позволяет хранить текстовые данные:

```
s = ''                  # пустая строка
s = 'name'              # строка из 4 символов
s = "name"              # та же строка из 4 символов
s = "('-')"             # кавычки создания строки могут хранить другие кавычки

s = r"""                # храним больше одной строки (ASCII art)
      ,~~.
     (  9 )-_,
(\___ )=='-'
 \ .   ) )
  \ `-' /
   `~j-'
     "=:
"""
```

Типы данных: строка (string)

Позволяет хранить текстовые данные:

```
s = ''                  # пустая строка
s = 'name'              # строка из 4 символов
s = "name"              # та же строка из 4 символов
s = "('-')"             # кавычки создания строки могут хранить другие кавычки
```

Основы строк:

- Использование управляющих последовательностей
- Доступ к произвольному элементу строки – индексация, срез
- Объединение строк – конкатенация, повторение
- Другие полезные и классные методы

Типы данных: строка (string)

Позволяет хранить текстовые данные:

```
s = ''                  # пустая строка
s = 'name'              # строка из 4 символов
s = "name"              # та же строка из 4 символов
s = "('-')"             # кавычки создания строки могут хранить другие кавычки
```

Основы строк:

- Использование управляющих последовательностей
- Доступ к произвольному элементу строки - индексация, срез
- Объединение строк - конкатенация, повторение
- Другие полезные и классные методы

Типы данных: строка (string)

Управляющие последовательности:

- специальные символы, которые не могут быть легко набраны на клавиатуре

Синтаксис:

```
"\n", "\t", "\'", "\"", "\\", ...
```

Пример:

1.
```
columns = "name\t|\tsurname\t|\tgroup\n"
row1 = "Elon\t|\tMask\t|\t201"
print(columns + row1)
# name    |       surname |       group
  Elon    |       Mask    |       201
```

Типы данных: строка (string)

Управляющие последовательности:

- специальные символы, которые не могут быть легко набраны на клавиатуре

Синтаксис:

```
"\n", "\t", "\'", "\"", "\\", ...
```

Пример:

```
2. print("¯\\_(ツ)_/¯")  # "¯\_(ツ)_/¯"
```

Типы данных: строка (string)

Управляющие последовательности:

- специальные символы, которые не могут быть легко набраны на клавиатуре

Синтаксис:

```
"\n", "\t", "\'", "\"", "\\", ...
```

Пример:

```
2. print("¯\\_(ツ)_/¯")  # "¯\_(ツ)_/¯"
3. print(r"¯\_(ツ)_/¯")  # "¯\_(ツ)_/¯"  - пример raw string
```

Типы данных: строка (string)

Позволяет хранить текстовые данные:

```
s = ''                  # пустая строка
s = 'name'              # строка из 4 символов
s = "name"              # та же строка из 4 символов
s = "('-')"             # кавычки создания строки могут хранить другие кавычки
```

Основы строк:

- Использование управляющих последовательностей
- Доступ к произвольному элементу строки – индексация, срез
- Объединение строк – конкатенация, повторение
- Другие полезные и классные методы

Типы данных: строка (string)

Индексация, срез:

- доступ к произвольному элементу строки, или срезу строки

Синтаксис:

```
s[i], s[i:j]  # результат - строка
```

Пример:

1.
```
models = [
    'A-152',
    'A-8731',
    'A-72'
]
models[0][2:]  # "152"
```

Типы данных: строка (string)

Позволяет хранить текстовые данные:

```
s = ''                  # пустая строка
s = 'name'              # строка из 4 символов
s = "name"              # та же строка из 4 символов
s = "('-')"             # кавычки создания строки могут хранить другие кавычки
```

Основы строк:

- Использование управляющих последовательностей
- Доступ к произвольному элементу строки – индексация, срез
- Объединение строк – конкатенация, повторение
- Другие полезные и классные методы

Типы данных: строка (string)

Конкатенация, повторение:

Синтаксис:

```
s1 + s2, s1*10
```

Пример:

1.
```
columns = "name\t|\tsurname\t|\tgroup\n"
row1 = "Elon\t|\tMask\t|\t201"
print(columns + row1)
# name    |       surname |       group
  Elon    |       Mask    |       201
```

Типы данных: строка (string)

Конкатенация, повторение:

Синтаксис:

```
s1 + s2, s1*10
```

Пример:

2.
```
progress = 0.42
bar_length = 30
completed = int(progress*bar_length)
progress_bar = "[" + "#"*completed + " "*(bar_length - completed) + "]"
print(progress_bar, progress*100 + "%")
# [############                  ] 42.0%
```

Типы данных: строка (string)

Позволяет хранить текстовые данные:

```
s = ''                  # пустая строка
s = 'name'              # строка из 4 символов
s = "name"              # та же строка из 4 символов
s = "('-')"             # кавычки создания строки могут хранить другие кавычки
```

Основы строк:

- Использование управляющих последовательностей
- Доступ к произвольному элементу строки – индексация, срез
- Объединение строк – конкатенация, повторение
- Другие полезные и классные методы

Типы данных: строка (string)

Метод .join():

Синтаксис:

```
separator.join(iterable)
''.join([1, 2, 3]
```

Пример:

```
1. columns = ['name', 'surname', 'group']
   row1 = ['Elon', 'Mask', '201']
   separator = '\t|\t'
   print(separator.join(columns))          # name    |       surname |       group
   print(separator.join(row1))               Elon    |       Mask    |       201
```

Типы данных: строка (string)

Форматирование строки:

- https://pyformat.info/

Синтаксис:

```
'{} {}'.format(variable1, variable2)
f'{variable1} {variable2}'
```

Пример:

1.
```
progress = 0.42
bar_length = 30
completed = int(progress*bar_length)
progress_bar = '[{}{}]'.format('#'*completed, ' '*(bar_length - completed))
print(progress_bar, progress*100 + "%")  # [############                  ] 42.0%
```

Типы данных: строка (string)

Разбиение строки:

- разбить строку в список по определенной подстроке

Синтаксис:

```
string.split(substring)
'1, 2, 3'.split(',')        # [1, 2, 3]
```

Пример:

```
1. models = [
     'A-152',
     'A-8731’
   ]
   models[0].split('-')  # ['A', '152']
```

Типы данных: строка (string)

Разбиение строки:

- разбить строку в список по определенной подстроке

Синтаксис:

```
string.split(substring)
'1, 2, 3'.split(',')        # [1, 2, 3]
```

Пример:

```
2. text = "name\t|\tsurname\t|\tgroup\nElon\t|\tMask\t|\t201“
   rows = text.split('\n’)
   row[0].split('\t|\t')  # ['name', 'surname', 'group']
```

Типы данных: строка (string)

Создание	' ',\" \", \"\"\"\" \"\"\", str()	
Индексация, срез	s[i], s[i:j]	
Конкатенация, повторение	s1 + s2, s*3	
Методы l.method(..)	Поиск	s.find(substr)
	Форматирование	s.format()
	Разбиение	s.split(substr)
	Замена	s.replace(from, to)
Изменяемость	неизменяемый тип данных	

Типы данных: список (list)

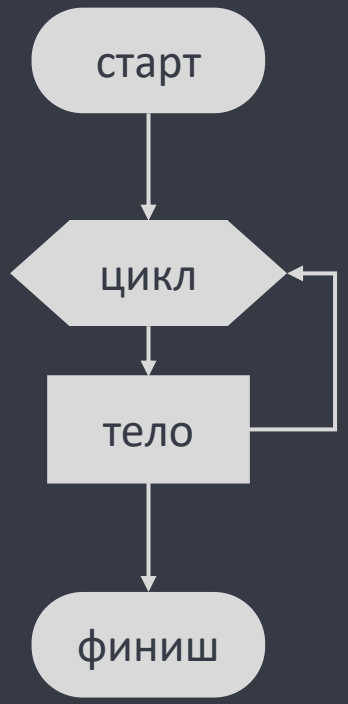
План на занятие:

1. VS Code text editor
2. Основные итерируемые объекты в Python и их возможности
   - 2.1. Список (list)
   - 2.2. Строка (str)
3. Итерации
   - 3.1. For .. in ..
   - 3.2. While ..
4. Логические выражения, алгебра логики (?)

Типы данных: строка (string)

Итерации и циклы

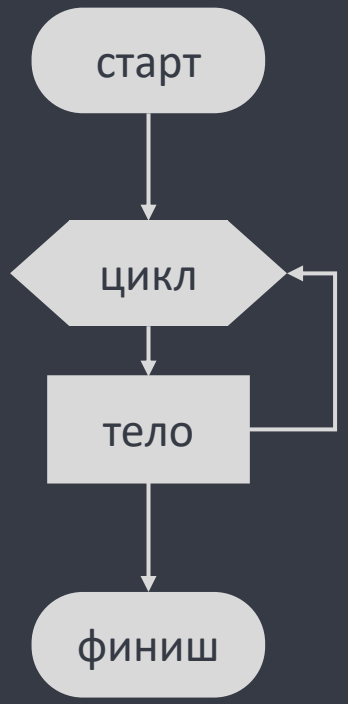
- многократное повторение действия

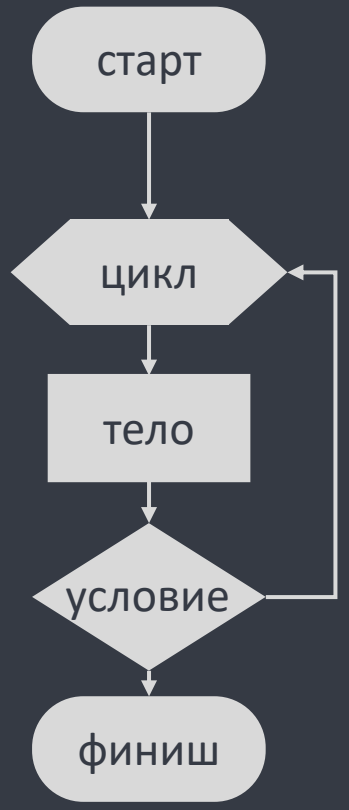


Типы данных: строка (string)

Итерации и циклы

- многократное повторение действия





Типы данных: строка (string)

Цикл for:

- позволяет итерироваться по элементам итерируемого объекта

Синтаксис:

```
for item in iterable:
    ____...
```

Пример:

```
1. models = ['A-152', 'A-8731', 'A-72']
   for model in models:
       print(model[2:])
   # '152'
     '8731'
     '72'
```

Типы данных: строка (string)

Цикл for:

- позволяет итерироваться по элементам итерируемого объекта

Синтаксис:

```
for item in iterable:
    ____...
```

Пример:

```
models = ['A-152', 'A-8731', 'A-72']
for i in range(3):
    print(model[i][2:])
# '152'
  '8731'
  '72'
```

Типы данных: строка (string)

Цикл for:

- позволяет итерироваться по элементам итерируемого объекта

Синтаксис:

```
for item in iterable:
    ____...
```

Пример:

```
3. models = ['A-152', 'A-8731', 'A-72']
   for i in range(len(models)):
       print(model[i][2:])
   # '152'
     '8731'
     '72'
```

Типы данных: строка (string)

Цикл for:

- позволяет итерироваться по элементам итерируемого объекта

Синтаксис:

```
for item in iterable:
    ____...
```

Пример:

```
4. models = ['A-152', 'A-8731', 'A-72']
   for letter in models[0]:
       print(letter, end=' ')
   # A - 1 5 2
```

Типы данных: строка (string)

Цикл for:

- позволяет итерироваться по элементам итерируемого объекта

Синтаксис:

```
for item in iterable:
    ____...
```

Пример:

```
columns = ['name', 'surname', 'group']
rows = [
    ['Elon', 'Mask', '201'],
    ['Elon', 'Mask', '201'],
    ['Elon', 'Mask', '201']
]
```

Типы данных: строка (string)

Цикл for:

- позволяет итерироваться по элементам итерируемого объекта

Синтаксис:

```
for item in iterable:
    ____...
```

Пример:

```
5. print(separator.join(columns))
   for row in rows:
       print(separator.join(row))
```