



Python как первый язык

День 2

Формальные языки

Неформальный язык

Найди сумму всех четных чисел меньше 10-ти

Формальный язык

Неформальный язык

Найди сумму всех четных чисел меньше 10-ти

Формальный язык

```
0: result = 0
1: i = 0
2: while i <= 10:
3:     if not i % 2:
4:         result += i
```

Неформальный язык

Найди сумму всех четных чисел меньше 10-ти

Формальный язык

```
0: result = 0
1: i = 0
2: while i <= 10:
3:     if not i % 2:
4:         result += i
5:     i += 1
```

- Сохраним результат как 0.
- Возьмем некий i равный 0
- Пока i меньше или равно 10, сложить i с результатом в том случае, если результат деления i на 2 равно нулю, и после этого увеличить i на единицу

Неформальный язык

Найди сумму всех четных чисел меньше 10-ти

Формальный язык

```
0: result = 0
1: i = 0
2: while i <= 10:
3:     if not i % 2:
4:         result += i
5:     i += 1
```

- Сохраним результат как 0.
- Возьмем некий i равный 0
- Пока i меньше или равно 10, сложить i с результатом в том случае, если результат деления i на 2 равно нулю, и после этого увеличить i на единицу

Неформальный язык

Найди сумму всех четных чисел меньше 10-ти

Формальный язык

```
0: result = 0
1: i = 0
2: while i <= 10:
3:     if not i % 2:
4:         result += i
5:     i += 1
```

- Сохраним результат как 0.
- Возьмем некий i равный 0
- Пока i меньше или равно 10, сложить i с результатом в том случае, если результат деления i на 2 равно нулю, и после этого увеличить i на единицу

Неформальный язык

Найди сумму всех четных чисел меньше 10-ти

Формальный язык

```
0: result = 0
1: i = 0
2: while i <= 10:
3:     if not i % 2:
4:         result += i
5:     i += 1
```

- Сохраним результат как 0.
- Возьмем некий i равный 0
- Пока i меньше или равно 10, сложить i с результатом в том случае, если результат деления i на 2 равно нулю, и после этого увеличить i на единицу

Неформальный язык

Найди сумму всех четных чисел меньше 10-ти

Формальный язык

```
0: result = 0
1: i = 0
2: while i <= 10:
3:     if not i % 2:
4:         result += i
5:     i += 1
```

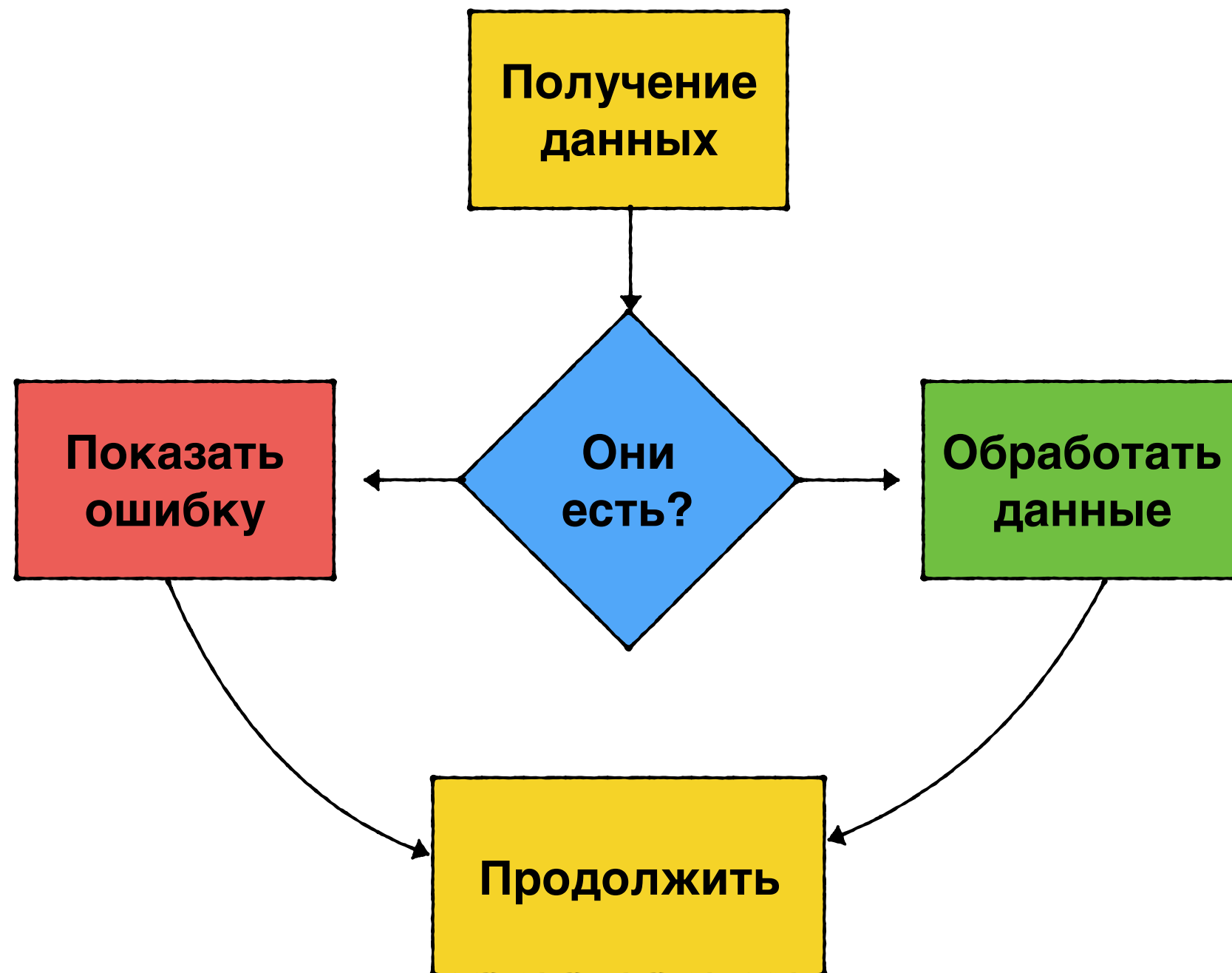
- Сохраним результат как 0.
- Возьмем некий i равный 0
- Пока i меньше или равно 10, сложить i с результатом в том случае, если результат деления i на 2 равно нулю, и после этого увеличить i на единицу

Программирование



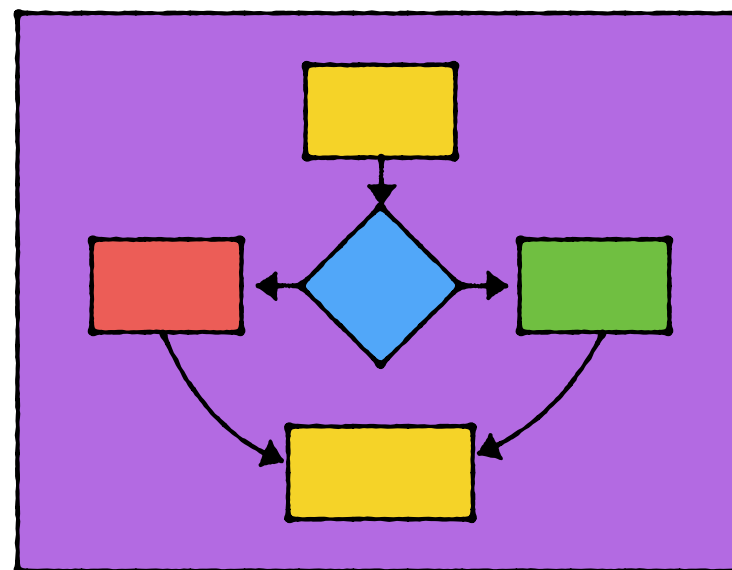
Построение алгоритмов

Построение алгоритмов



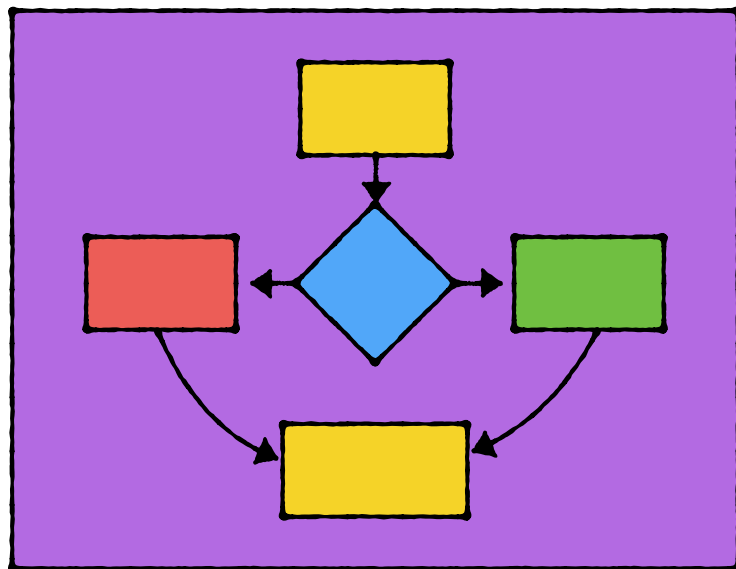
Построение алгоритмов

Функция

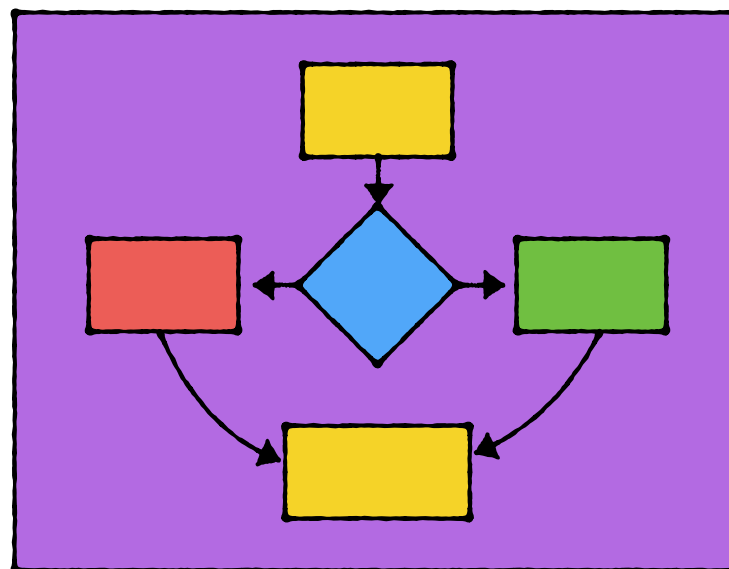


Построение алгоритмов

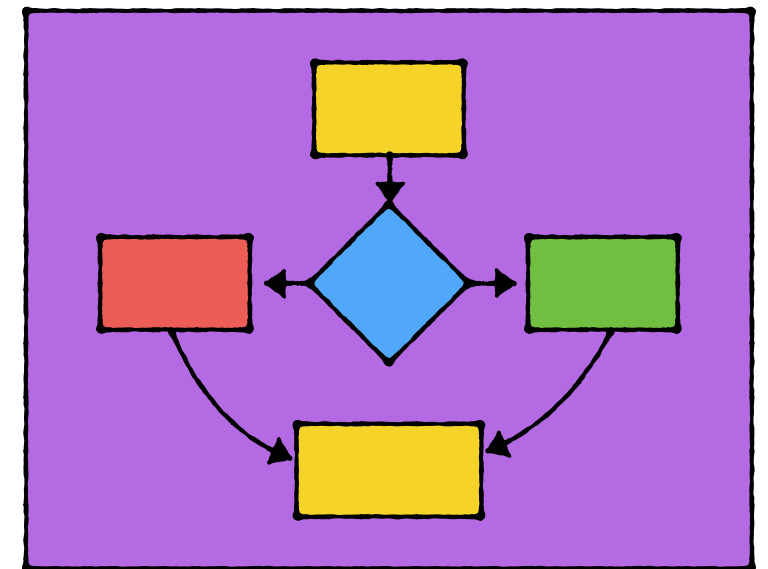
Функция



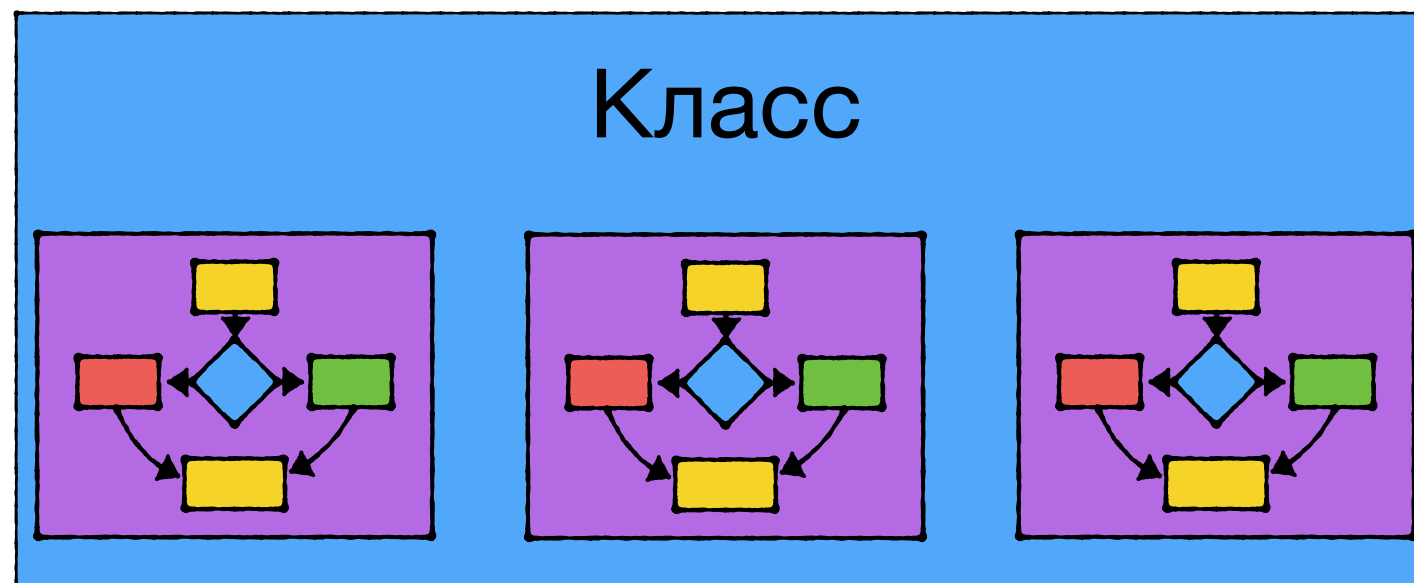
Функция



Функция

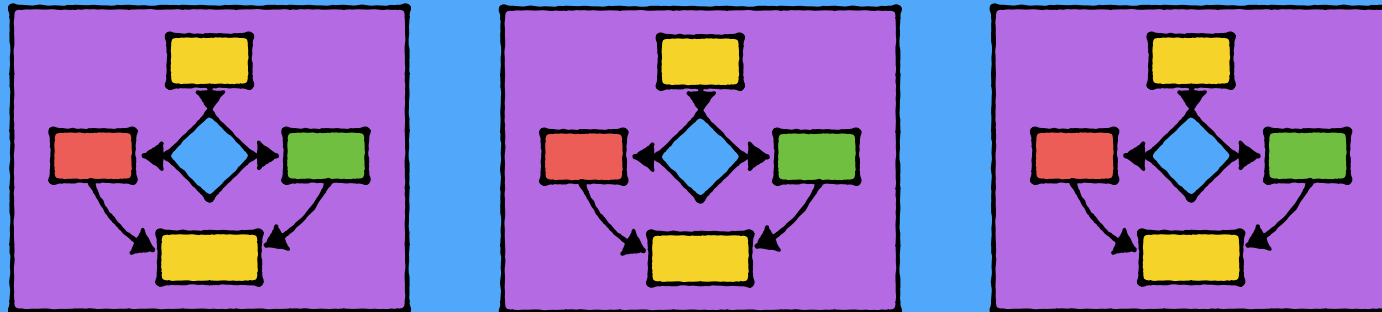


Построение алгоритмов

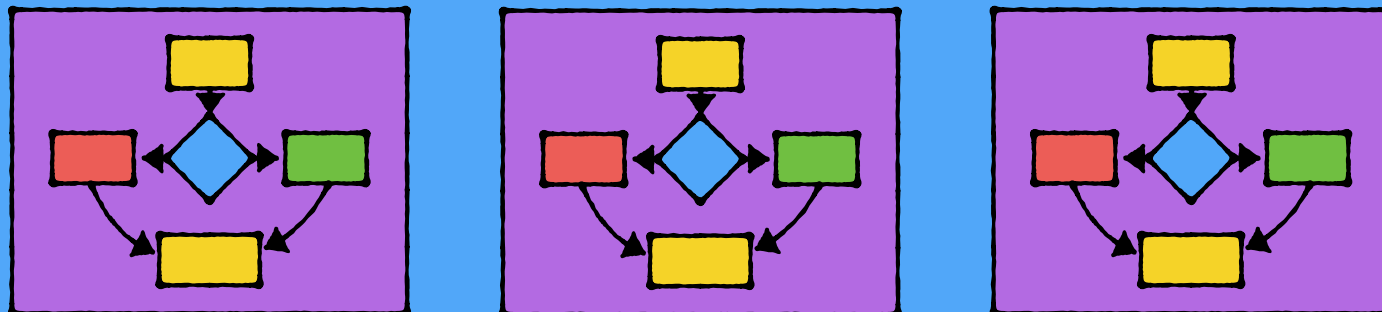


Построение алгоритмов

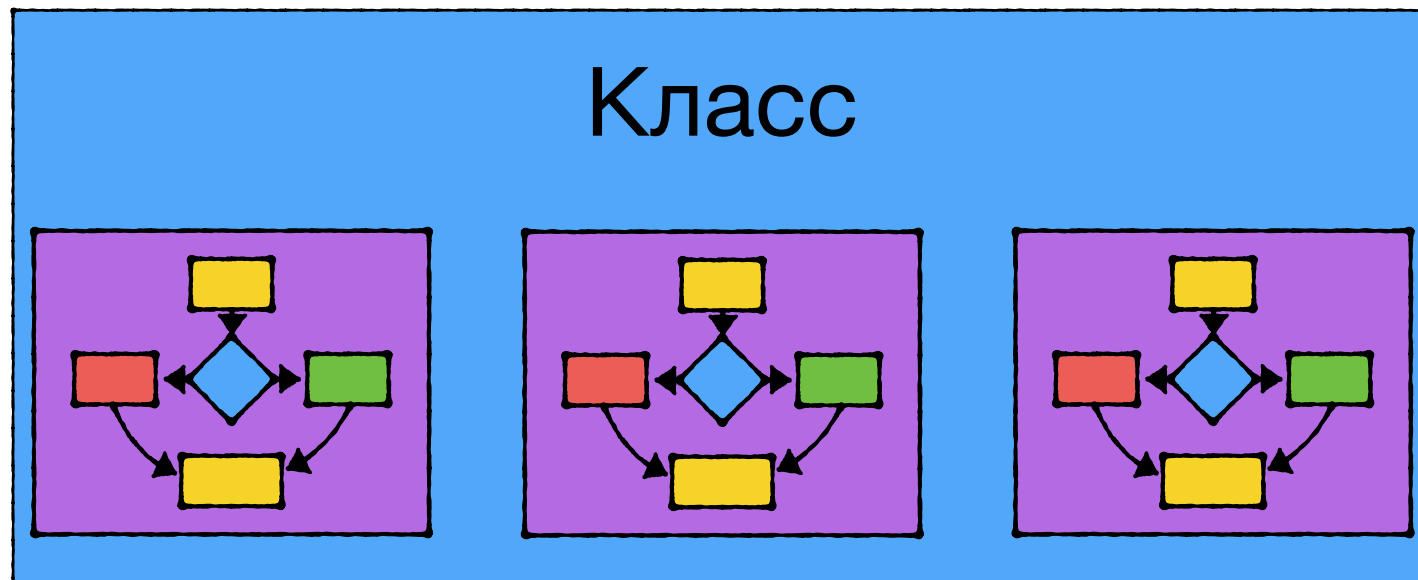
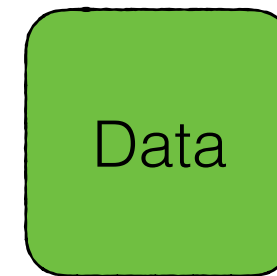
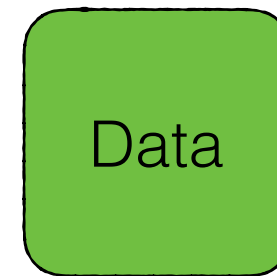
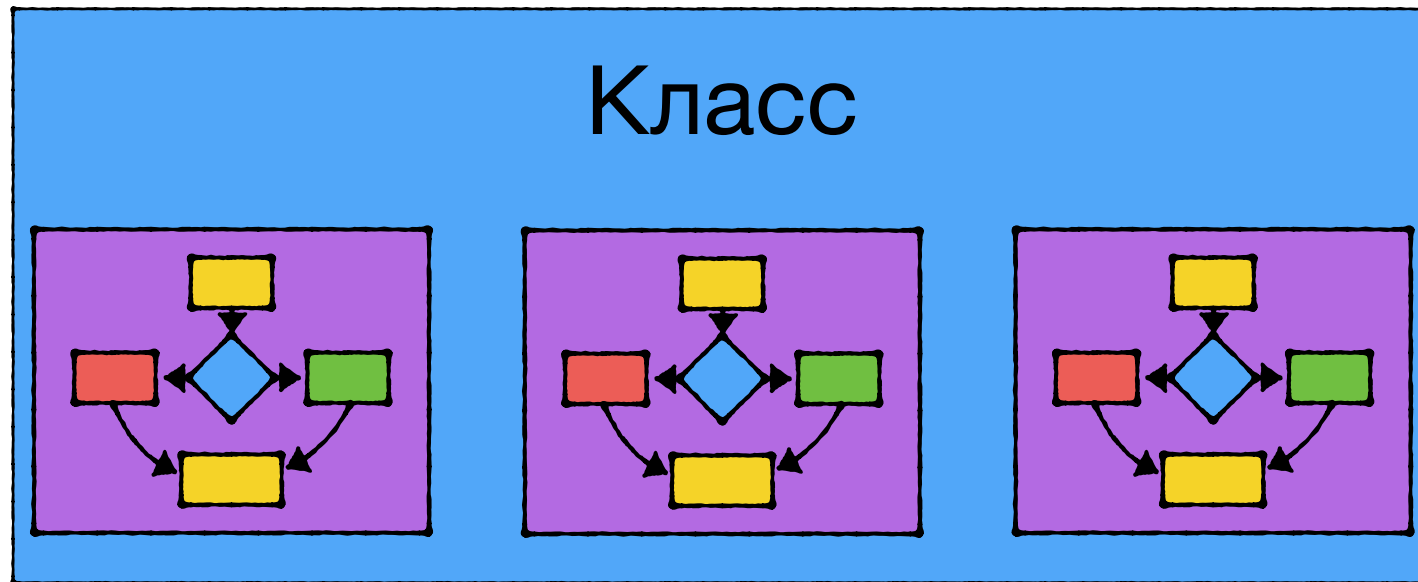
Класс



Класс

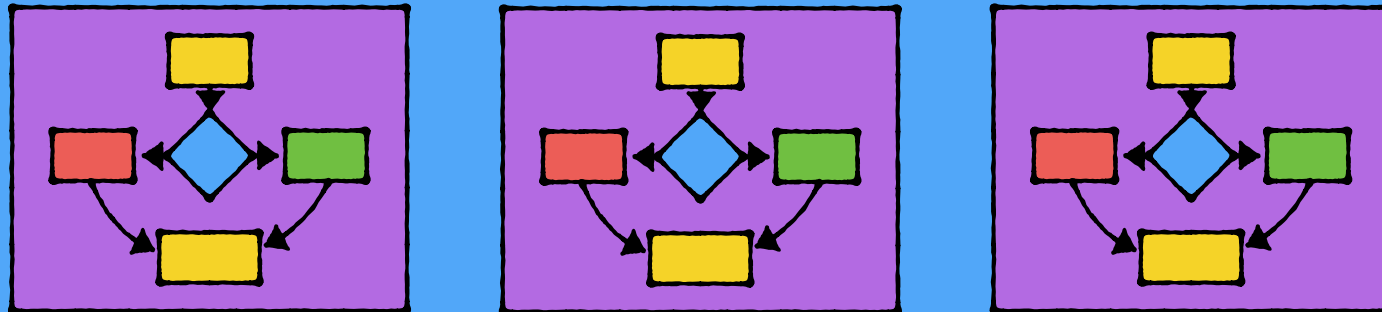


Построение алгоритмов

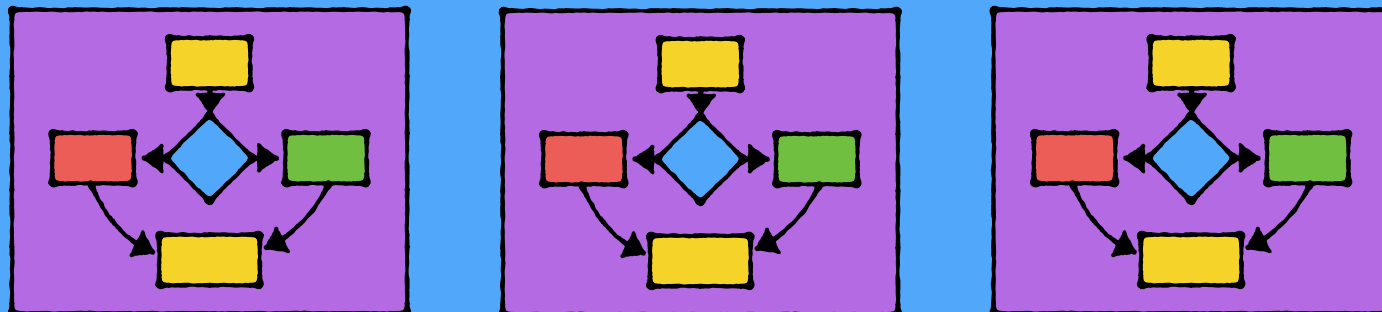


Построение алгоритмов

Класс



Класс

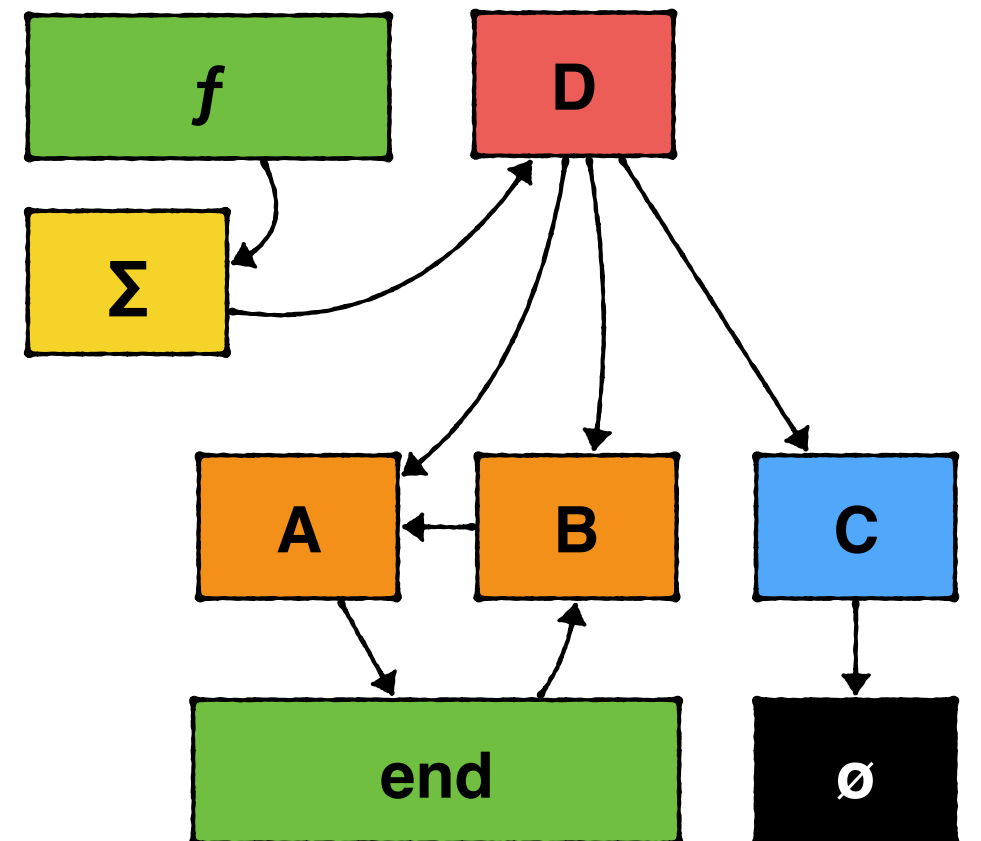


Data

Data

Data

Программа



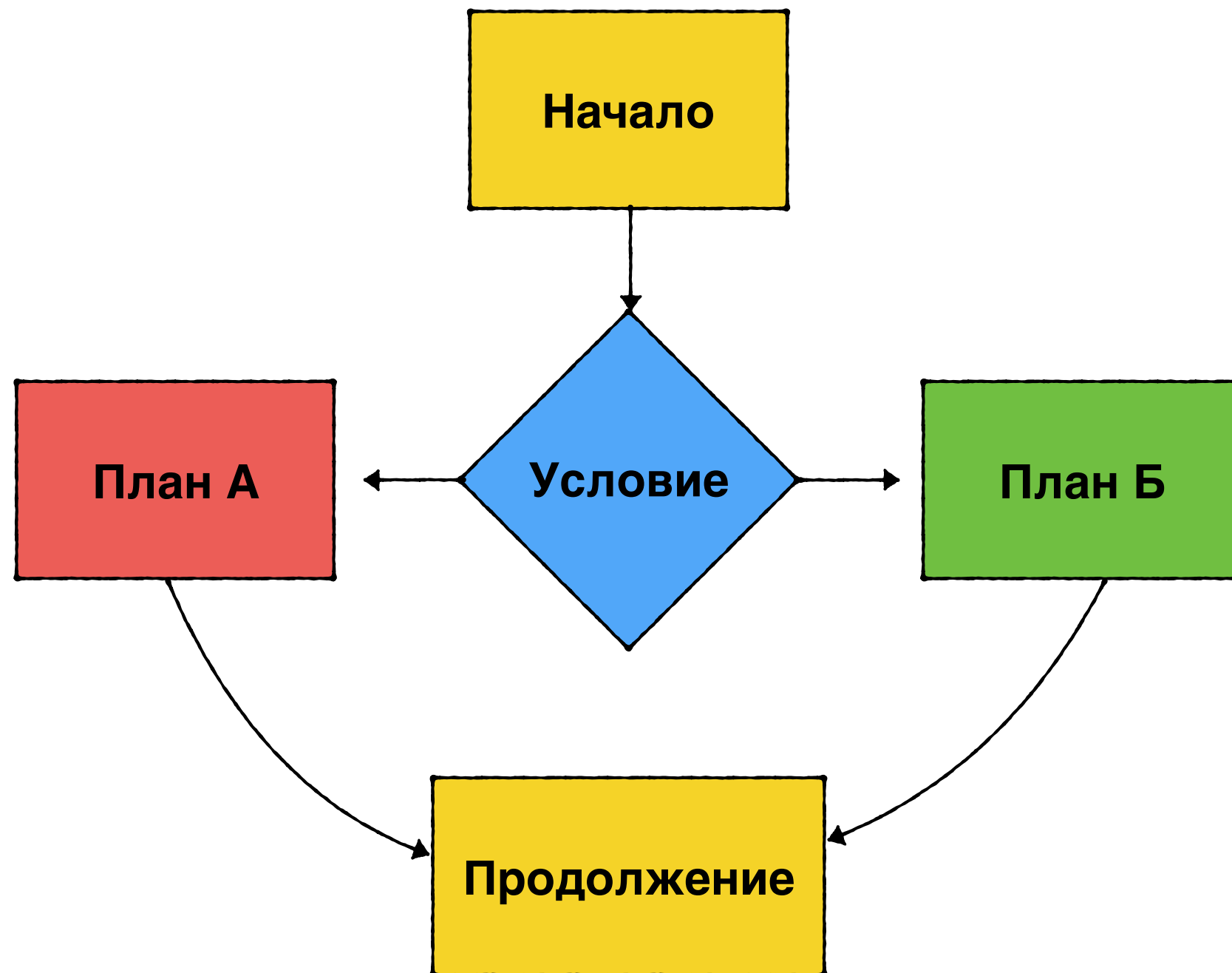
Управляющие конструкции

- Условные выражения
- Циклы
- Обработка ошибок

Условные выражения



Условные выражения



Условные выражения

```
0: if a == a:
1:     print("Все ок. Расходимся")
2: else:
3:     print("ZOMG! Этого не может быть!")
```

Условные выражения

```
0: if a > b:
1:     print("a больше b")
2: elif a == b:
3:     print("a в точности равно b")
4: else:
5:     print("a меньше b")
```

УСЛОВНЫЕ ВЫРАЖЕНИЯ

Тернарный оператор

```
0: res = "a больше b" if a > b else "a меньше b"
```

Условные выражения

Тернарный оператор

```
0: res = "a больше b" if a > b else "a меньше b"
```

значение

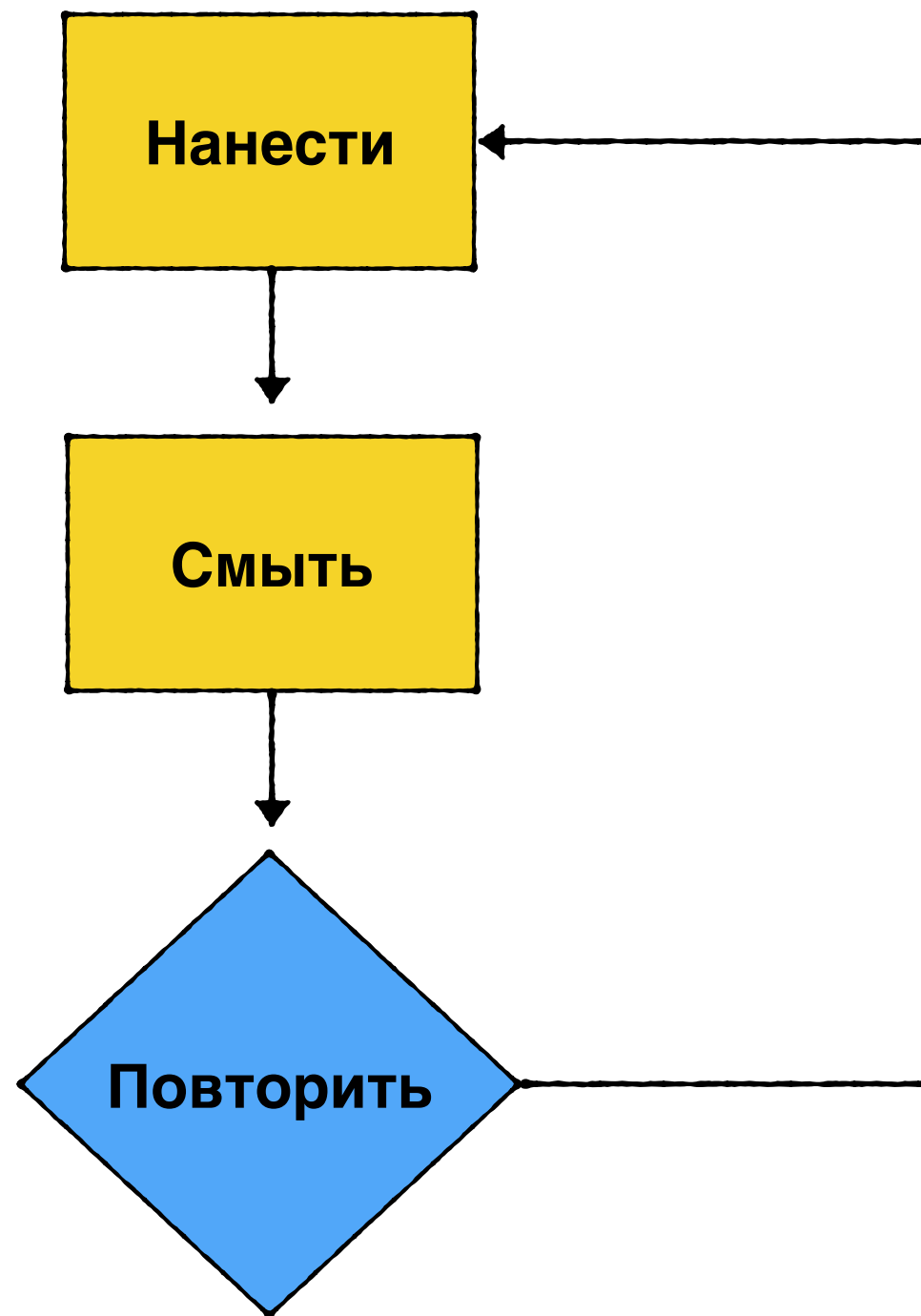
если выполняется

условие

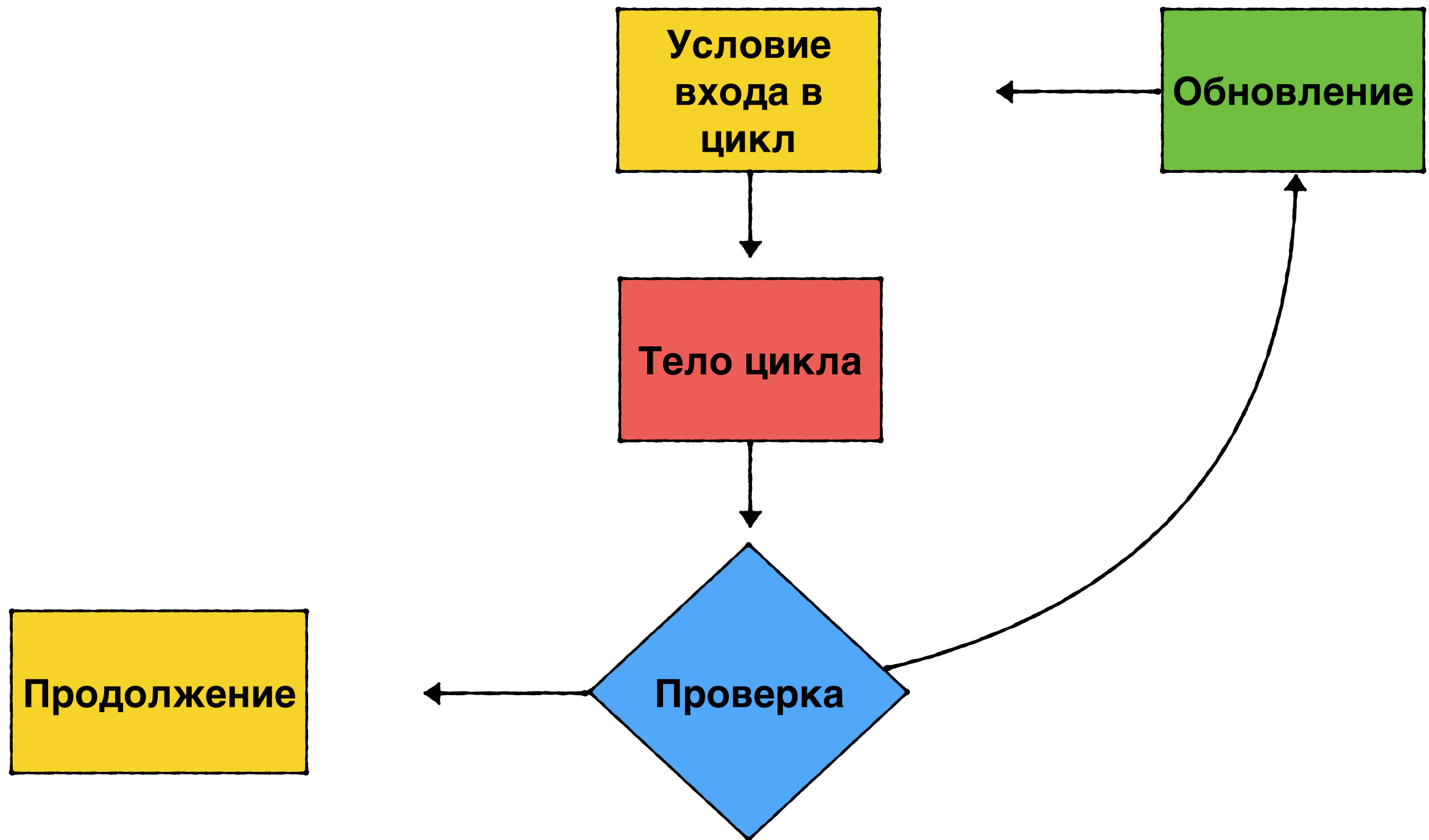
иначе

другое
значение

ЦИКЛЫ



ЦИКЛЫ



ЦИКЛЫ

Задача: найти сумму чисел от 1 до 1000

ЦИКЛЫ

Задача: найти сумму чисел от 1 до 1000

$$\theta: a = 1 + 2 + 3 + \dots + 1000$$

ЦИКЛЫ

Задача: найти сумму чисел от 1 до 1000

$$\emptyset: a = 1 + 2 + 3 + \dots + 1000$$



ЦИКЛЫ

while

```
0: while a < b:  
1:     do_something()  
2:     a = a + 1
```

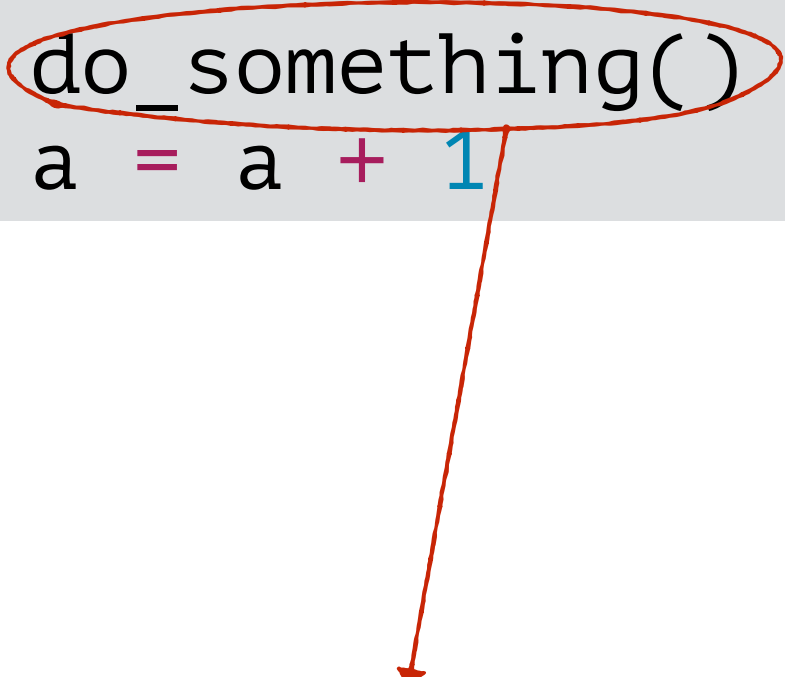


Условие

ЦИКЛЫ

while

```
0: while a < b:  
1:     do_something()  
2:     a = a + 1
```

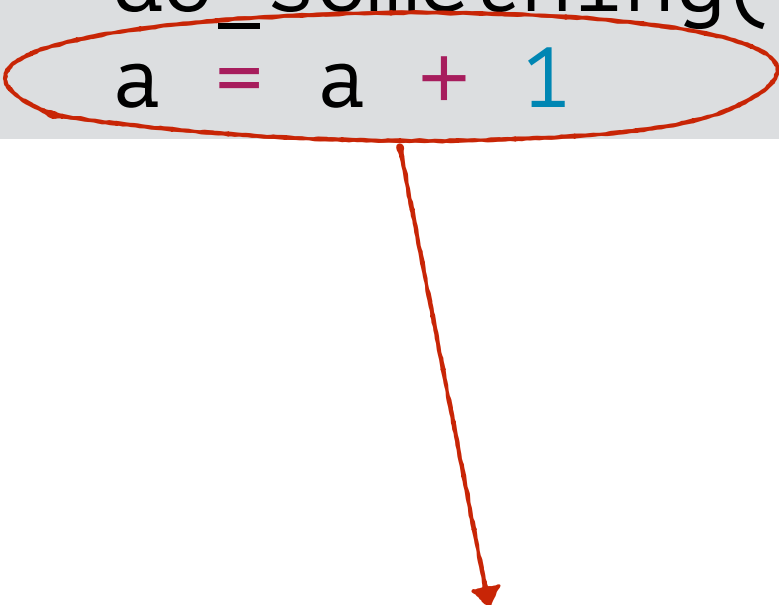


Итерация

ЦИКЛЫ

while

```
0: while a < b:  
1:     do_something()  
2:     a = a + 1
```



Обновление

ЦИКЛЫ

Задача: найти сумму чисел от 1 до 1000

```
0: i = 1
1: result = 0
2:
3: while i <= 1000:
4:     result = result + i
5:     i += 1
6:
7: print(result)
```

ЦИКЛЫ

Задача: найти сумму чисел от 1 до 1000

```
0: a = sum(range(1, 1001))
```

ЦИКЛЫ

for

```
0: for letter in "whoop a doop":  
1:     print(letter)
```

ЦИКЛЫ

for

```
0: for letter in "whoop a doop":  
1:     print(letter)
```

w
h
o
o
p

a

d
o
o
p

Циклы

for

```
0: for letter in "whoop a doop":  
1:     print(letter)
```

w
h
o
o
p

a

d
o
o
p

Циклы

for

```
0: for letter in "whoop a doop":  
1:     print(letter)
```

w
h
o
o
p

a

d
o
o
p

Циклы

for

```
0: for letter in "whoop a doop":  
1:     print(letter)
```

w
h
o
o
p

a

d
o
o
p

ЦИКЛЫ

for

Перечислимый тип

```
0: for letter in "whoop a doop":  
1:     print(letter)
```

w
h
o
o
p

a

d
o
o
p

Обработка ошибок

```
>>> 1 + '1'
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: unsupported operand type(s) for +:  
'int' and 'str'
```

Обработка ошибок

```
>>> 1 + '1'
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: unsupported operand type(s) for +:  
'int' and 'str'
```



Тип ошибки

Обработка ошибок

```
>>> 1 + '1'
```

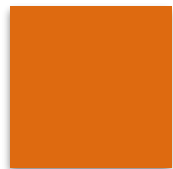
```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: unsupported operand type(s) for +:  
'int' and 'str'
```



Тип ошибки



Сообщение ошибки

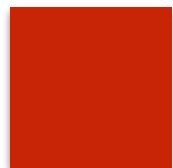
Обработка ошибок

```
>>> 1 + '1'
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: unsupported operand type(s) for +:  
'int' and 'str'
```



Тип ошибки



Сообщение ошибки



Traceback ошибки

Обработка ошибок

```
01: try:
01:     test = 1 + "1"
02: except TypeError:
02:     print("You can't add number to string")
```

Обработка ошибок

– общий случай

```
01: try:
02:     #совершить что-то сомнительное
03: except TypeError:
04:     #обработать TypeError
05: except ValueError:
06:     #обработать ValueError
07: except <какое-то другое исключение>:
08:     #обработать какое-то другое исключение
09: else:
10:     #если исключения не было
11: finally:
12:     #выполнить это в любом случае
```

```
while True:
    try:
        a = int(input('10 + 20 = ?\n'))
    except ValueError:
        print('answer should be an integer')
    else:
        if (a == 30):
            print('correct!')
            break;
        else:
            print('you are wrong!')
```

Структуры данных

CHUCK



GROUND KOOPA



KOOPA KUBES



MUSHROOM STEAK



SEVEN BONE



CHUCK ROAST



BRISKET

RIB



RIB ROAST



RIBEYE ROAST



BOWSER RIBS

CARRAPACE



KOOPA SOUP

RUMP



TOP RUMP



RUMP TIP



BOTTOM RUMP



EYE OF RUMP

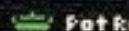


HOOPA

It's What's for Supper



Stew



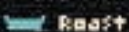
Pot Roast



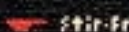
Skillet



Braise



Roast



Stir-Fry



Grill or Broil



Marinate & Grill or Broil

PAIRS WELL WITH:



Mushrooms



Carnivorous Vegetables



Red Wine

BREASTPLATE



SKIRT STEAK

KING'S LOIN



TRI-TIP



T-BONE

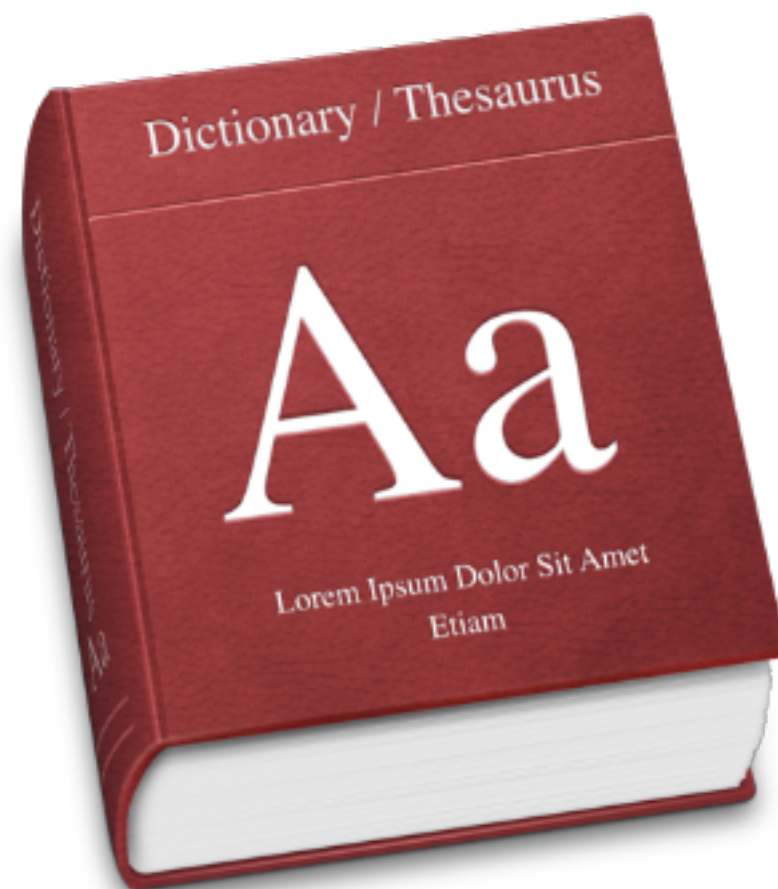


CROWN LOIN



TENDERLOIN

Структуры данных



Структуры данных

Список

```
0: names = ["Alex", "Johnny", "Fonzey"]
```

Структуры данных

Список

```
0: names = ["Alex", "Johnny", "Fonzey"]
```

Кортеж

```
0: names = ("Alex", "Johnny", "Fonzey")
```

Структуры данных

Список

```
0: names = ["Alex", "Johnny", "Fonzey"]
```

Кортеж

```
0: names = ("Alex", "Johnny", "Fonzey")
```

Словарь, хэш, объект

```
0: about = {  
1:     "first_name": "John",  
2:     "second_name": "Doe"  
3: }
```

Структуры данных

Множество

```
0: numbers = {"odd", "even"}
```

Последовательности

Enumerable types (перечислимые типы)

```
0: names = ["Alex", "Johnny", "Fonzey"]  
1: letters = "ABCDEFGH"
```

Последовательности

Enumerable types (перечислимые типы)

```
0: names = ["Alex", "Johnny", "Fonzey"]  
1: letters = "ABCDEFGH"  
2:  
3: names[0] # -> "alex"
```


Последовательности

Enumerable types (перечислимые типы)

```
0: names = ["Alex", "Johnny", "Fonzey"]
1: letters = "ABCDEFGH"
2:
3: names[0] #-> "alex"
3: letters[0] #-> "A"
```

Последовательности

Enumerable types (перечислимые типы)

```
0: names = ["Alex", "Johnny", "Fonzey"]
1: letters = "ABCDEFG"
2:
3: names[0] #-> "alex"
3: letters[0] #-> "A"
```

Индексация

```
0: names = ["Alex", "Johnny", "Fonzey"]
```


первый элемент
нулевой индекс
names[0]

Последовательности

Enumerable types (перечислимые типы)

```
0: names = ["Alex", "Johnny", "Fonzey"]
1: letters = "ABCDEFG"
2:
3: names[0] #-> "Alex"
3: letters[0] #-> "A"
```

Индексация

```
0: names = ["Alex", "Johnny", "Fonzey"]
```



второй элемент
первый индекс
names[1]

Последовательности

Enumerable types (перечислимые типы)

```
0: names = ["Alex", "Johnny", "Fonzey"]
1: letters = "ABCDEFG"
2:
3: names[0] #-> "Alex"
3: letters[0] #-> "A"
```

Индексация

```
0: names = ["Alex", "Johnny", "Fonzey"]
```


третий элемент
второй индекс
names[2]

Последовательности

Индексация

```
>>> "abcdefg"[0]  
'a'
```

Последовательности

Индексация

```
>>> "abcdefg"[0]  
'a'
```

a	b	c	d	e	f	g
0	1	2	3	4	5	6

Последовательности

Индексация

```
>>> "abcdefg"[0]  
'a'  
>>> "abcdefg"[-1]  
'g'
```

Последовательности

Индексация

```
>>> "abcdefg"[0]
'a'
>>> "abcdefg"[-1]
'g'
```

a	b	c	d	e	f	g
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Последовательности

Индексация

```
>>> "abcedfg"[0]
'a'
>>> "abcdefg"[-1]
'g'
>>> "abcdefg"[0:3] #срез от 0 до 3 (не включительно)
'abc'
```

Последовательности

Индексация

```
>>> "abcdefg"[0]
'a'
>>> "abcdefg"[-1]
'g'
>>> "abcdefg"[0:3] #срез от 0 до 3 (не включительно)
'abc'
```

a	b	c	d	e	f	g
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Последовательности

Индексация

```
>>> "abcdefg"[0]
'a'
>>> "abcdefg"[-1]
'g'
>>> "abcdefg"[0:3] #срез от 0 до 3 (не включительно)
'abc'
```

a	b	c	d	e	f	g
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Последовательности

Индексация

“abcdefg”[0:3]

a	b	c	d	e	f	g
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Последовательности

Индексация

“abcdefg”[3:5]

a	b	c	d	e	f	g
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Последовательности

Индексация

“abcdefg”[-4:-1]

a	b	c	d	e	f	g
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Работа со списками и кортежами

Работа со списками и кортежами

Добавление элемента в конец

```
0: names = ["Alex", "Johnny", "Fonzey"]  
1: names + ["Donatello"]
```


Работа со списками и кортежами

Добавление элемента в конец

```
0: names = ["Alex", "Johnny", "Fonzey"]  
1: names + ["Donatello"]
```

```
['Alex', 'Johnny', 'Fonzey', 'Donatello']  
>>>
```

Работа со списками и кортежами

Добавление элемента в конец

```
0: names = ["Alex", "Johnny", "Fonzey"]  
1: names + ["Donatello"]
```

```
['Alex', 'Johnny', 'Fonzey', 'Donatello']  
>>>
```

```
0: names = ["Alex", "Johnny", "Fonzey"]  
1: names.append("Donatello")
```

Работа со списками и кортежами

Добавление элемента в конец

```
0: names = ["Alex", "Johnny", "Fonzey"]  
1: names + ["Donatello"]
```

```
['Alex', 'Johnny', 'Fonzey', 'Donatello']  
>>>
```

```
0: names = ["Alex", "Johnny", "Fonzey"]  
1: names.append("Donatello")
```

```
>>>  
>>> names  
['Alex', 'Johnny', 'Fonzey', 'Donatello']
```

Работа со списками и кортежами

Добавление элемента в конец

```
0: names = ["Alex", "Johnny", "Fonzey"]  
1: names + ["Donatello"]
```

```
0: names = ("Alex", "Johnny", "Fonzey")  
1: names + ("Donatello",)
```

Работа со списками и кортежами

Изменение элементов

```
0: names = ["Alex", "Johnny", "Fonzey"]  
1: names[0] = ["Donatello"]
```

Работа со списками и кортежами

Изменение элементов

```
0: names = ["Alex", "Johnny", "Fonzey"]  
1: names[0] = ["Donatello"]
```

```
>>> names  
['Donatello', 'Johnny', 'Fonzey']
```

Работа со списками и кортежами

Изменение элементов

```
0: names = ["Alex", "Johnny", "Fonzey"]  
1: names[0] = "Donatello"
```

```
>>> names  
['Donatello', 'Johnny', 'Fonzey']
```

```
0: names = ("Alex", "Johnny", "Fonzey")  
1: names[0] = "Donatello"
```

Работа со списками и кортежами

Изменение элементов

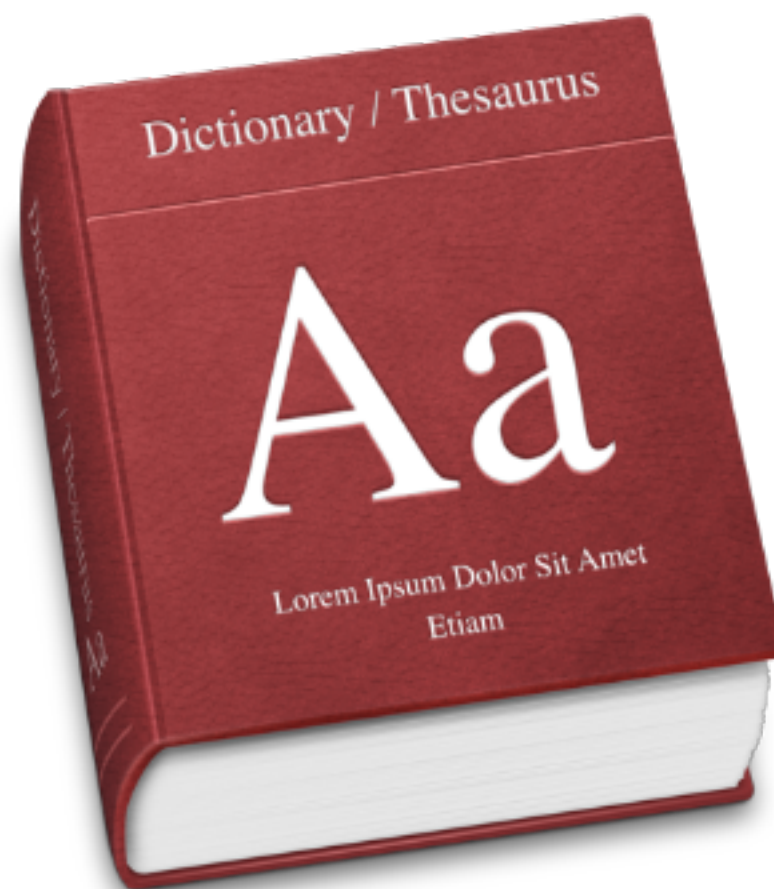
```
0: names = ["Alex", "Johnny", "Fonzey"]  
1: names[0] = "Donatello"
```

```
>>> names  
['Donatello', 'Johnny', 'Fonzey']
```

```
0: names = ("Alex", "Johnny", "Fonzey")  
1: names[0] = "Donatello"
```

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: 'tuple' object does not support item  
assignment
```


Словари (dicts)



Словари (dicts)

Создание словаря

```
0: about = {  
1:     "first_name": "Alex",  
2:     "second_name": "Schroeder",  
3:     "age": "28"  
4: }
```

Словари (dicts)

Создание словаря

```
0: about = {  
1:     "first_name": "Alex",  
2:     "second_name": "Schroeder",  
3:     "age": "28"  
4: }
```

first_name	Alex
second_name	Schroeder
age	28

Словари (dicts)

Создание словаря

```
0: about = {  
1:     "first_name": "Alex",  
2:     "second_name": "Schroeder",  
3:     "age": "28"  
4: }
```

Получение свойства

```
0: about["first_name"]
```

Мутабельность

Свойство типов данных быть измененными

Мутабельность

Свойство типов данных быть измененными

```
>>> s = "hello"
```

```
>>> s[0] = 10
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: 'str' object does not support item assignment
```

Мутабельность

Свойство типов данных быть измененными

```
>>> s = "hello"  
>>> s[0] = 10  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: 'str' object does not support item assignment
```

Строки в Python иммутабельны

Мутабельность

Свойство типов данных быть измененными

```
>>> l = [1, 2, 3]
>>> l[0] = 10
>>> l
[10, 2, 3]
```


Мутабельность

Свойство типов данных быть измененными

```
>>> t = (1, 2, 3)
```

```
>>> t[0] = 10
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: 'tuple' object does not support item assignment
```

Кортежи в Python иммутабельны

Ссылки

```
0: list_1 = [1, 2, 3]
1: list_2 = [1, 2, 3]
2:
3: list_1 == list_2 # True
4: list_1 is list_2 # False
```

Ссылки

```
0: list_1 = [1, 2, 3]
1: list_2 = [1, 2, 3]
2:
3: list_1 == list_2 # True
4: list_1 is list_2 # False
5:
6: 1 == 1 # True
7: 1 is 1 # True
```

input
output



Вывод

```
0: print("Hello, World!")
```

```
>>> print("Hello, World!")  
'Hello, World'
```

Вывод

```
0: print("Hello, World!")
```

ВВОД

```
0: name = input("Your name\n")
```

Немного о спецсимволах

```
0: print("Хочу разбить это на две строчки")
```



Немного о спецсимволах

```
0: print("Хочу разбить это на две строки")
```

```
0: print("""Хочу разбить  
0: это на две строки""")
```

Немного о спецсимволах

```
0: print("Хочу разбить это на две строки")
```

```
0: print("Хочу разбить\nэто на две строки")
```

Немного о спецсимволах

```
0: print("Хочу разбить это на две строчки")
```

```
0: print("Хочу разбить\nэто на две строчки")
```

спецсимвол перевода строки

Немного о спецсимволах

```
0: print("Хочу использовать "кавычки")
```

Немного о спецсимволах

```
0: print("Хочу использовать "кавычки"")
```

```
File "<stdin>", line 1
    print("Хочу использовать "кавычки"")
                              ^
```

```
SyntaxError: invalid syntax
```

Немного о спецсимволах

```
0: print("Хочу использовать "кавычки"")
```

```
File "<stdin>", line 1
    print("Хочу использовать "кавычки"")
                              ^
```

```
SyntaxError: invalid syntax
```

```
0: print("Хочу использовать \"кавычки\"")
```

экранирование

Немного о спецсимволах

```
0: print("Хочу использовать "кавычки"")
```

```
File "<stdin>", line 1
    print("Хочу использовать "кавычки"")
                              ^
```

```
SyntaxError: invalid syntax
```

```
0: print("Хочу использовать \"кавычки\"")
```

экранирование

Немного о спецсимволах

```
0: print("Хочу использовать символ: \u2030")
```


Немного о спецсимволах

```
0: print("Хочу использовать символ: \u2030")
```

```
'Хочу использовать символ: %'
```