

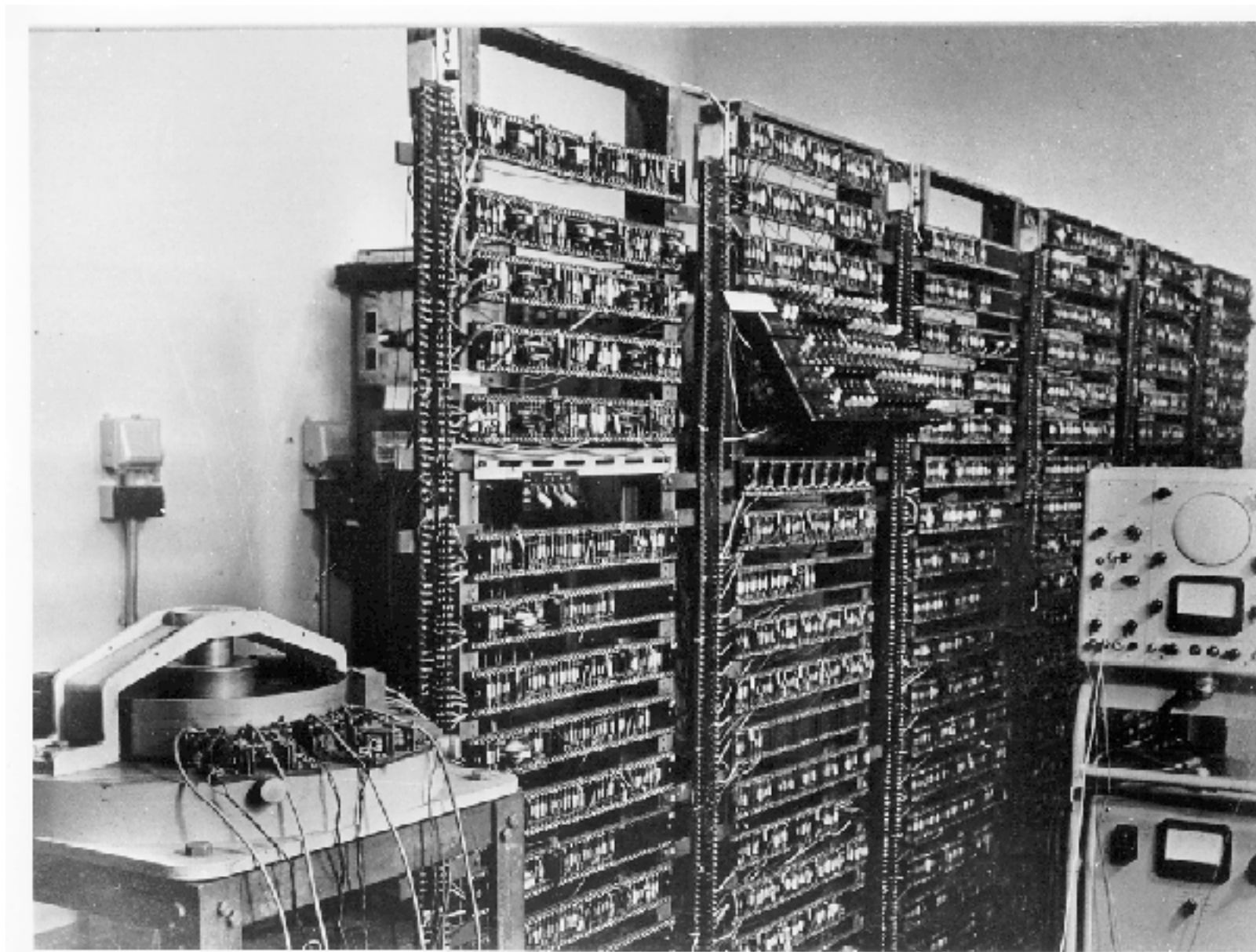


# Python как первый язык

День 1

Пару слов о программировании

# Компьютер - это набор транзисторов



Компьютер - это набор транзисторов

Программирование - это переключение транзисторов

0x001	0x002	0x003	0x004	0x005	0x006	0x007	0x008
1	0	0	1	1	0	1	0

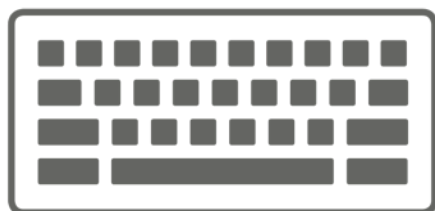


```
max(2, 3, 20, 123, 7, 12)
```

Как все это работает?

# Механизм выполнения

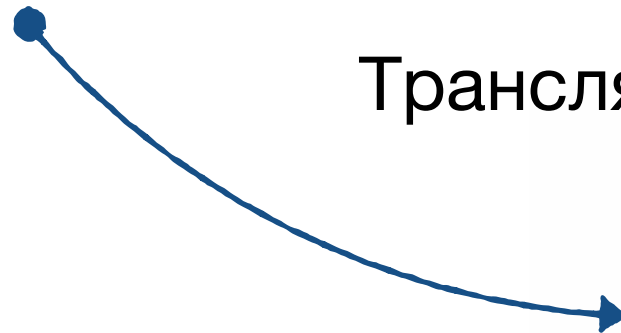
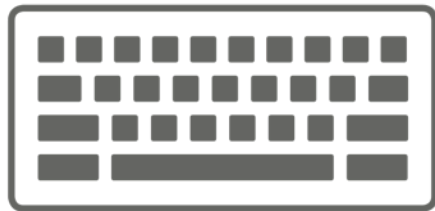
Файл или устройство ввода



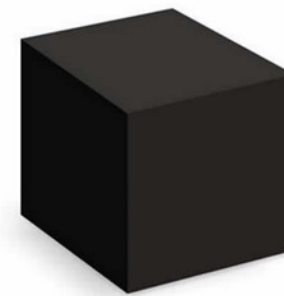


# Механизм выполнения

Файл или устройство ввода

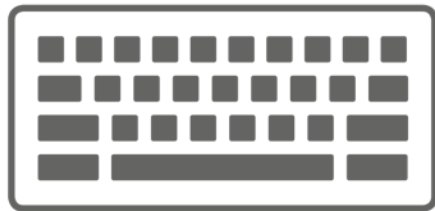


Транслятор в машинный код



# Механизм выполнения

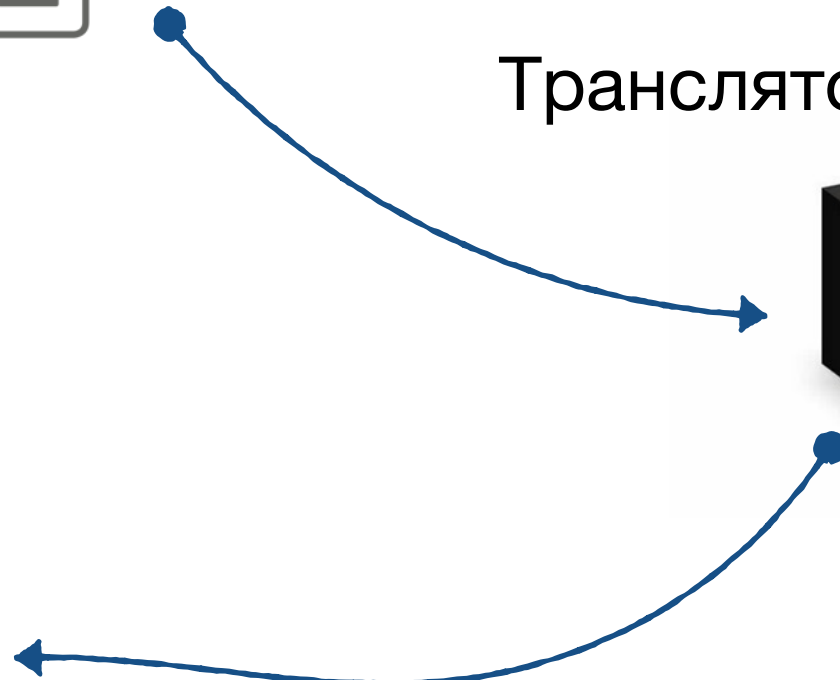
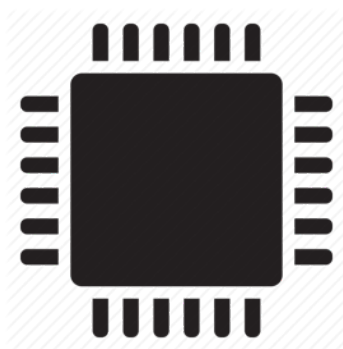
Файл или устройство ввода



Транслятор в машинный код

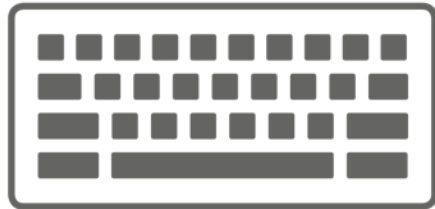


Выполнение



# Механизм выполнения

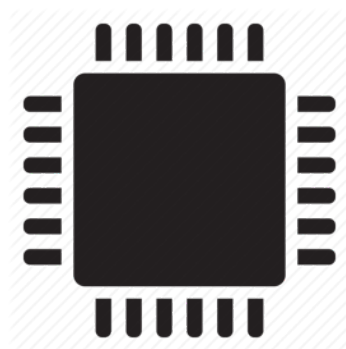
Файл или устройство ввода



Транслятор в машинный код



Выполнение



Файл или устройство вывода





# Python



Создан Гвидо ван Россумом

# Python



Создан Гвидо ван Россумом  
Актуальны версии: 2.x и 3.x

# Python



Создан Гвидо ван Россумом  
Актуальны версии: 2.x и 3.x  
Питон забавен

# Python

- Простой в использовании
- Мощный
- Востребованный



# Python

- Простой в использовании
- Мощный
- Востребованный
  
- Интерпретируемый
- Типизированный
- Переносимый
- Объектно-ориентированный

# Python

- Простой в использовании
- Мощный
- Востребованный
  
- Интерпретируемый
- Типизированный
- Переносимый
- Объектно-ориентированный
  
- И вообще, большой молодец

# В чем хорош Python?

- Анализ данных
- Консольные утилиты
- Веб-программирование
- Desktopные приложения
- Микросервисы

# В чем Python плох

- Конкурентное программирование
- Системное программирование (?)

# Starter Pack



Текстовый редактор Sublime Text 3  
(написан на Python)

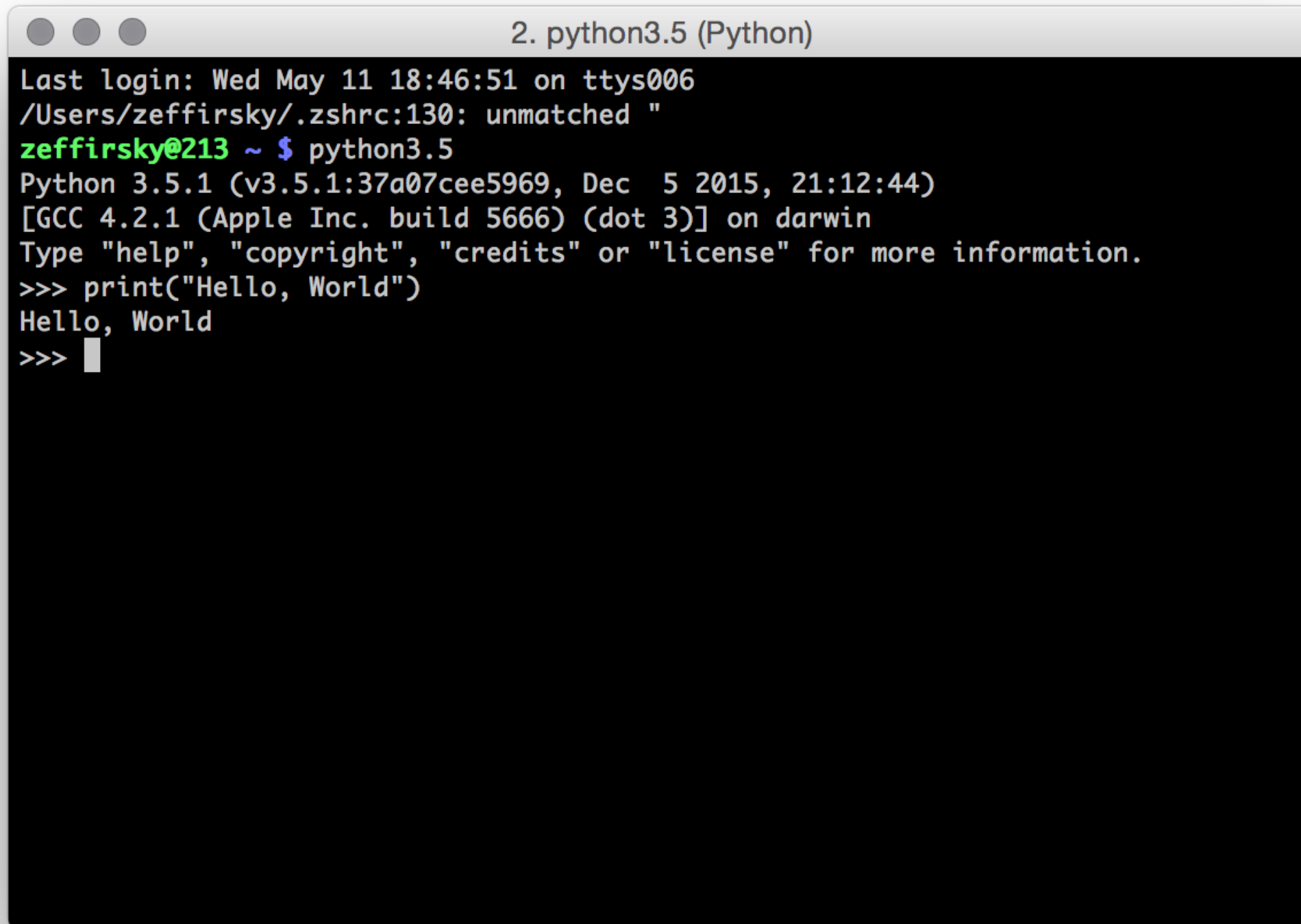


Консольная утилита  
(Terminal, iTerm, console, cygwin)



Интерпретатор Python

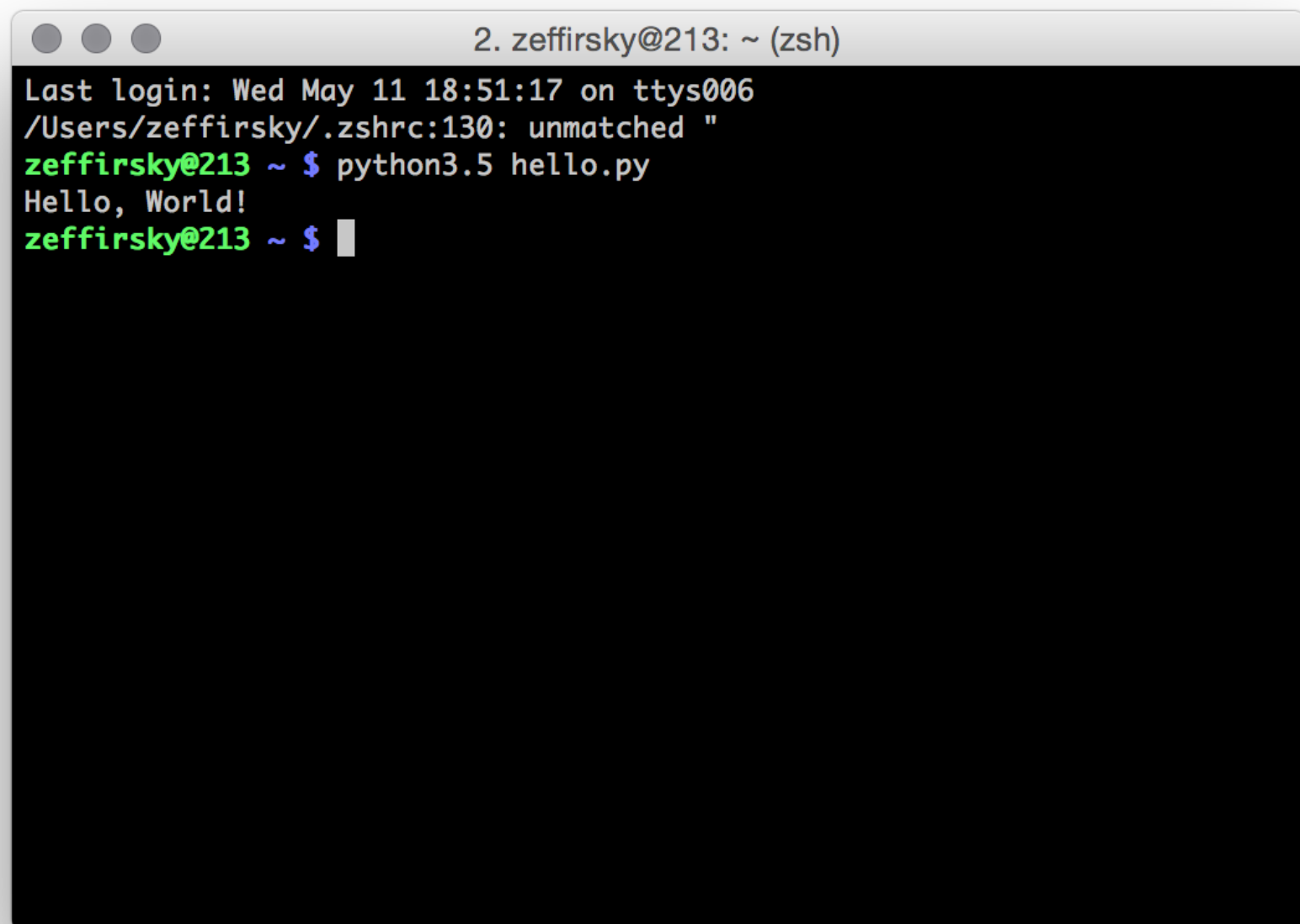
# Работа с Python через консоль



```
2. python3.5 (Python)
Last login: Wed May 11 18:46:51 on ttys006
/Users/zeffirsky/.zshrc:130: unmatched "
zeffirsky@213 ~ $ python3.5
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  5 2015, 21:12:44)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World")
Hello, World
>>> █
```

# Работа с .py файлами

```
file: hello.py  
0: print("Hello, World!")
```



A terminal window titled "2. zeffirsky@213: ~ (zsh)" with three window control buttons in the top-left corner. The terminal output shows the login process, the execution of the command `python3.5 hello.py`, and the resulting output "Hello, World!". The prompt `zeffirsky@213 ~ $` is followed by a cursor.

```
2. zeffirsky@213: ~ (zsh)  
Last login: Wed May 11 18:51:17 on ttys006  
/Users/zeffirsky/.zshrc:130: unmatched "  
zeffirsky@213 ~ $ python3.5 hello.py  
Hello, World!  
zeffirsky@213 ~ $ █
```

# Синтаксис

- Грамматика языка
- Набор ключевых слов
- Правила формирования наименований

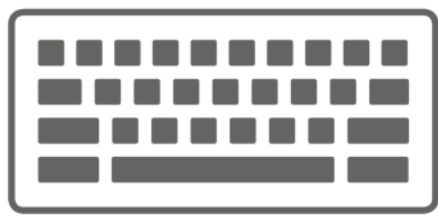


# Синтаксис

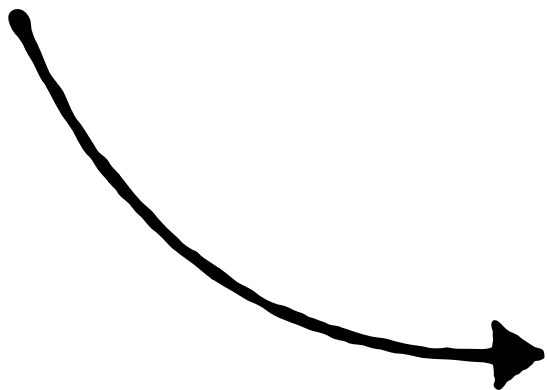
- Грамматика языка
- Набор ключевых слов
- Правила формирования наименований
- Соглашения и правила хорошего тона
- Хорошие и плохие практики

# СИНТАКСИС

Файл: hello.py



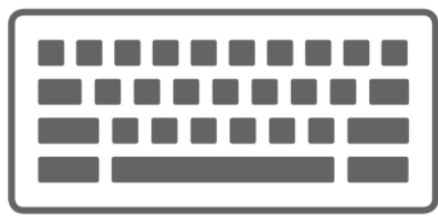
```
0: def say_hello(name):  
1:     print("Hello " + name)  
2:  
3: my_name = "Trdat"  
3:  
3: say_hello(my_name)
```



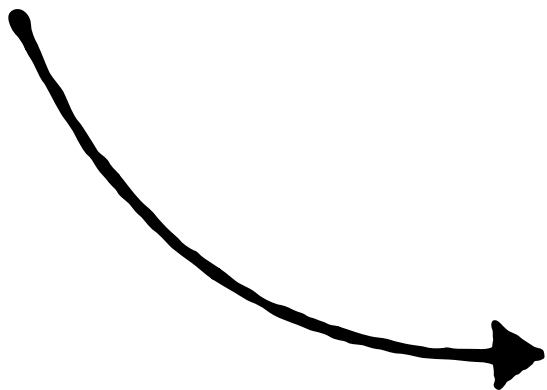
Интерпретатор python

# Синтаксис

Файл: hello.py



```
0: def say_hello(name):  
1:     print("Hello " + name)  
2:  
3: my_name = "Trdat"  
3:  
3: say_hello(my_name)
```



Интерпретатор python

- Текст превращается в AST
- AST компилируется в .рус байт-код
- Выполняется байт-код

# AST

Абстрактное синтаксическое дерево

# AST

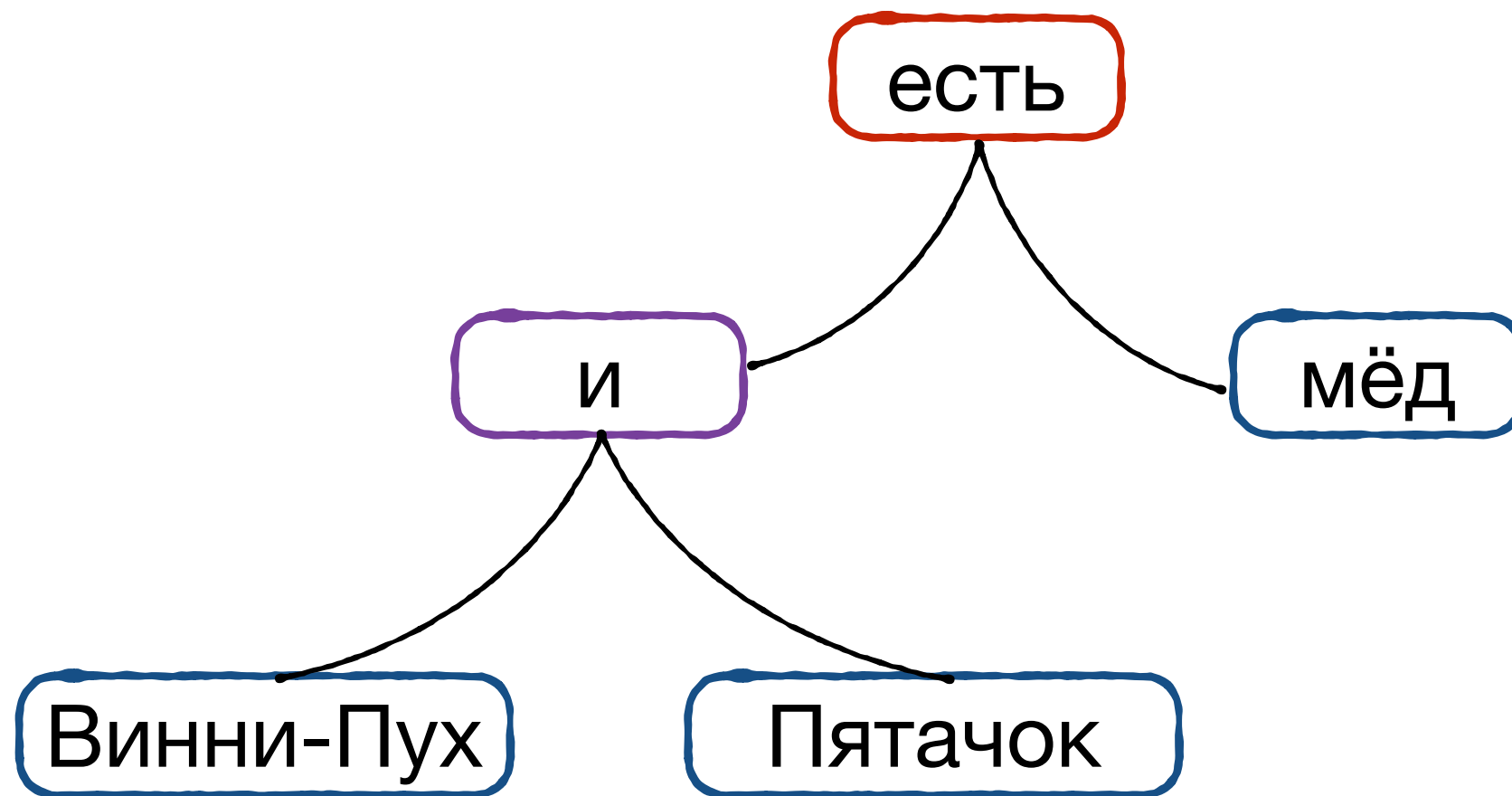
Абстрактное синтаксическое дерево

Винни-Пух и Пятачок едят мёд

# AST

Абстрактное синтаксическое дерево

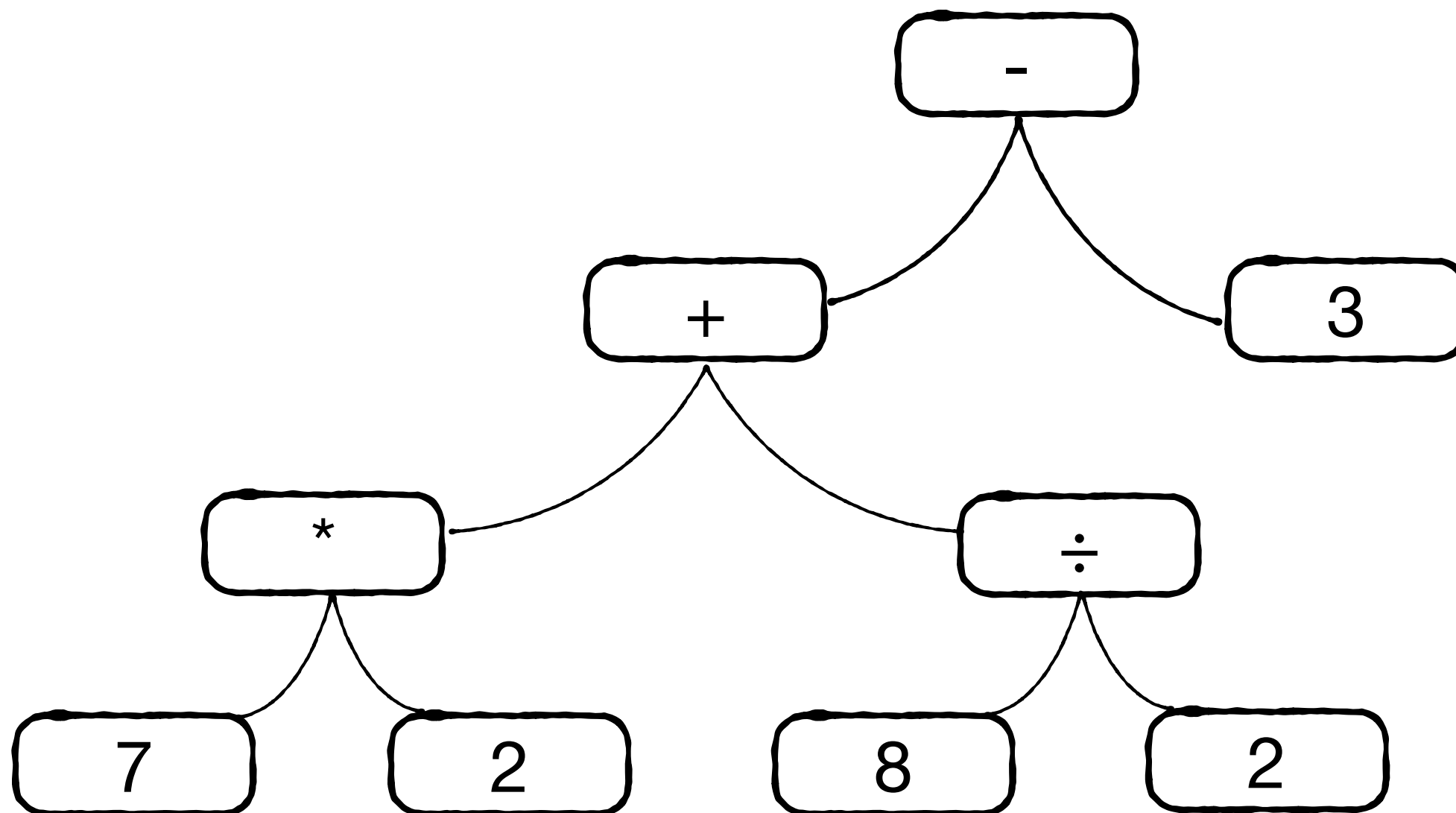
Винни-Пух и Пятачок едят мёд



# AST

Абстрактное синтаксическое дерево

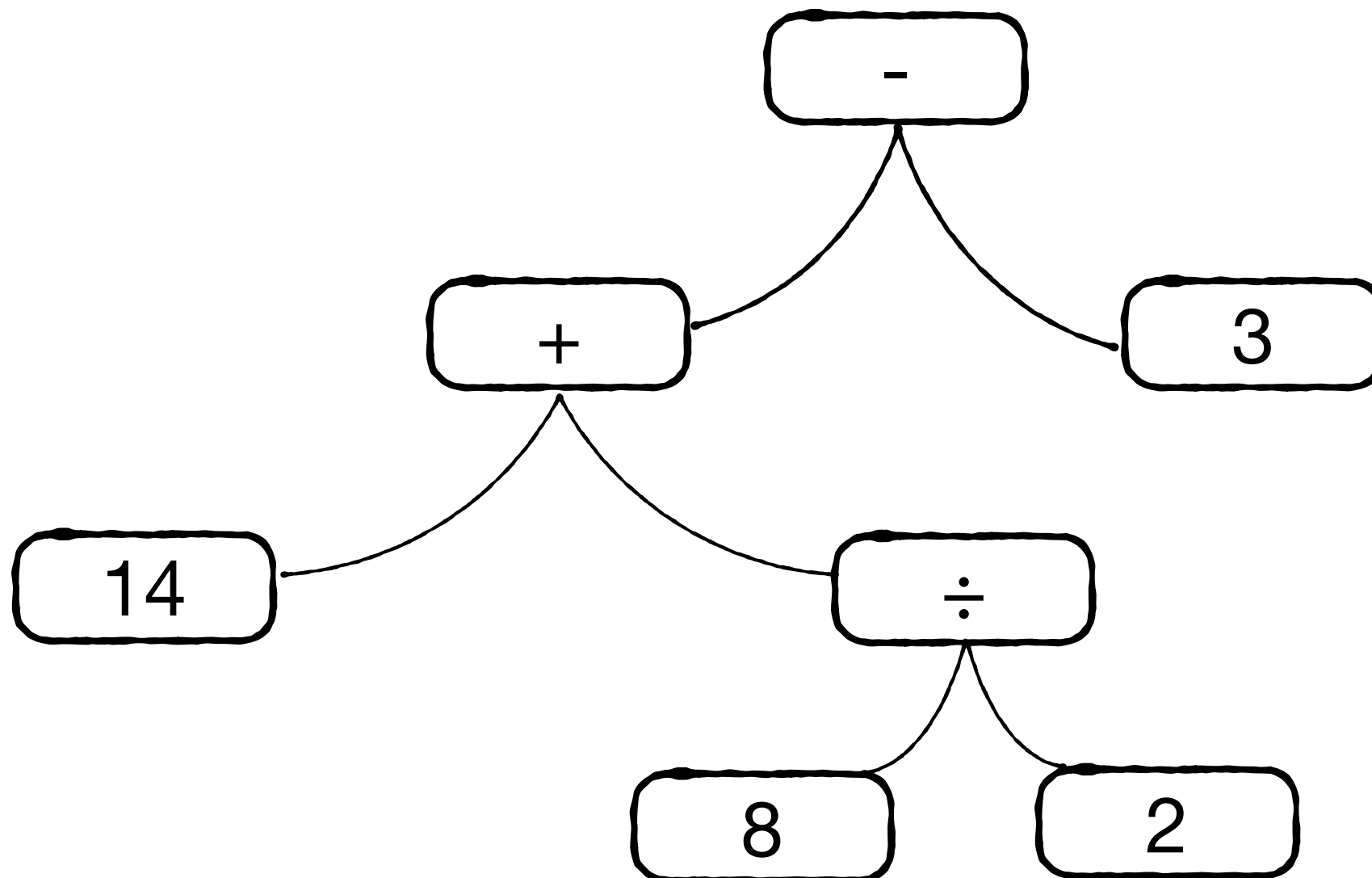
$3 + 7 * 2 - 8 \div 2$



# AST

Абстрактное синтаксическое дерево

$3 + 7 * 2 - 8 \div 2$

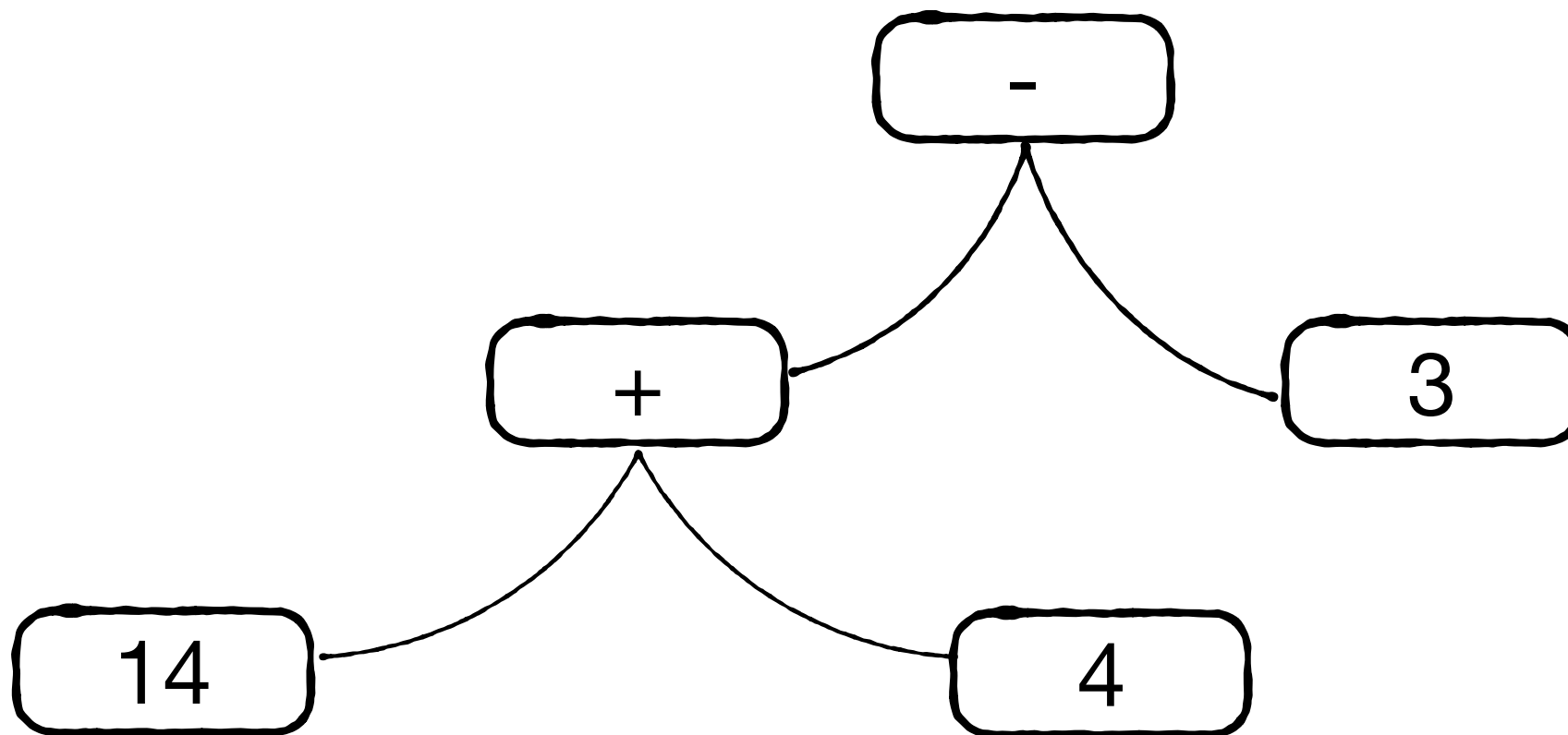




# AST

Абстрактное синтаксическое дерево

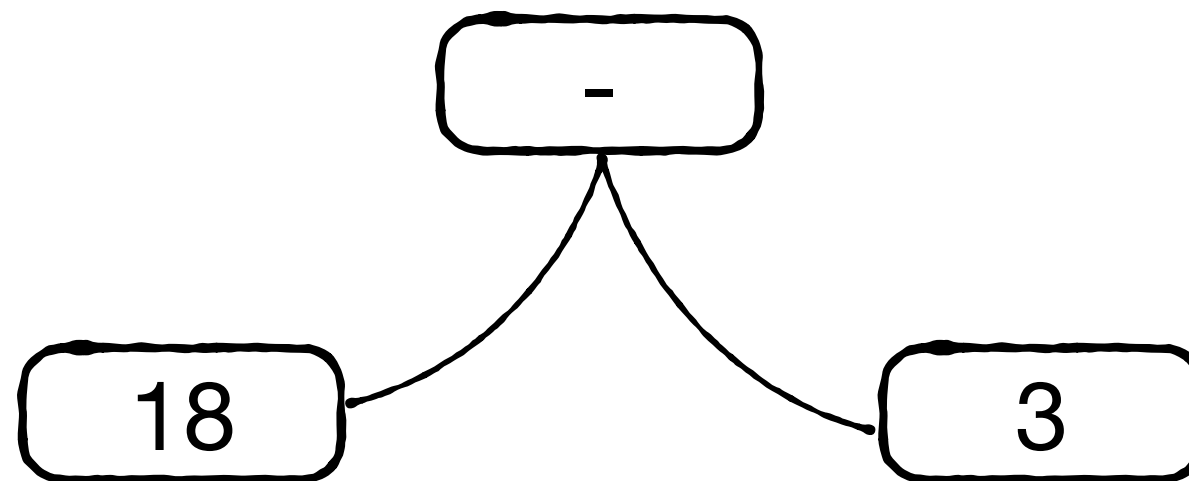
$3 + 7 * 2 - 8 \div 2$



# AST

Абстрактное синтаксическое дерево

$3 + 7 * 2 - 8 \div 2$



# AST

Абстрактное синтаксическое дерево

$3 + 7 * 2 - 8 \div 2$

15

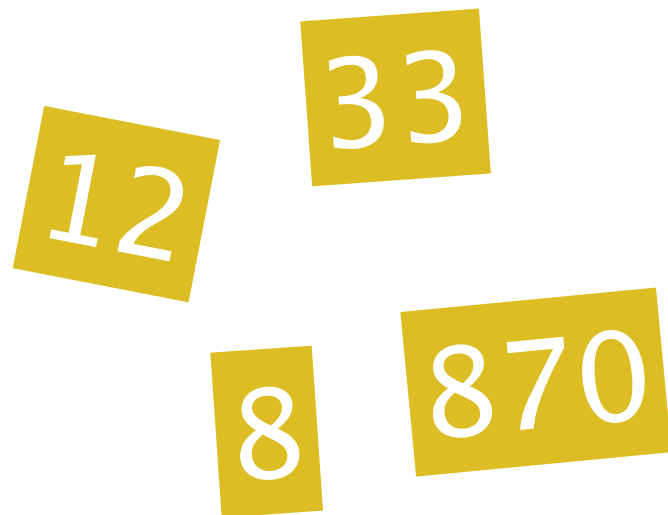


# Типы данных

– множество значений и операций на этих значениях

# Типы данных

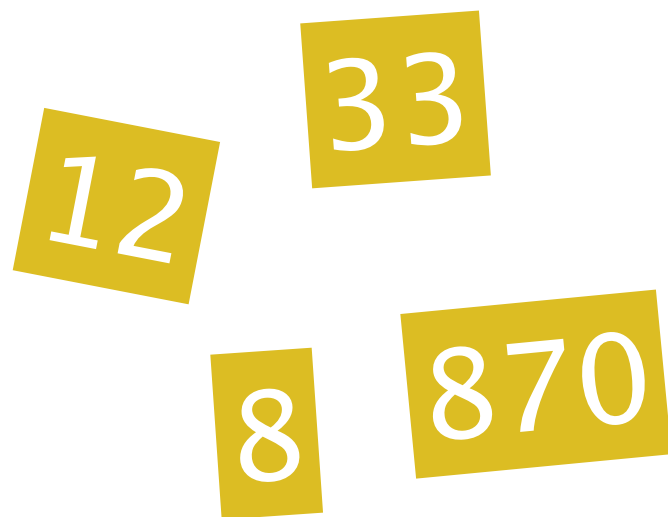
– множество **значений** и операций на этих значениях



Числа

# Типы данных

– множество **значений** и операций на этих значениях



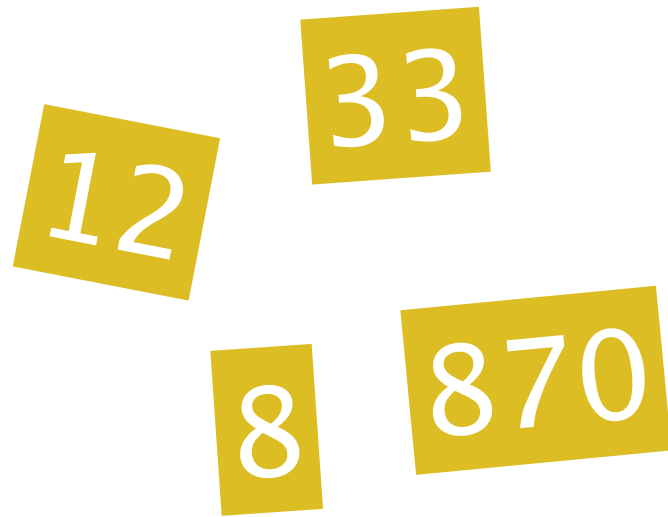
Числа



Буквы

# Типы данных

– множество значений и **операций** на этих значениях



Числа

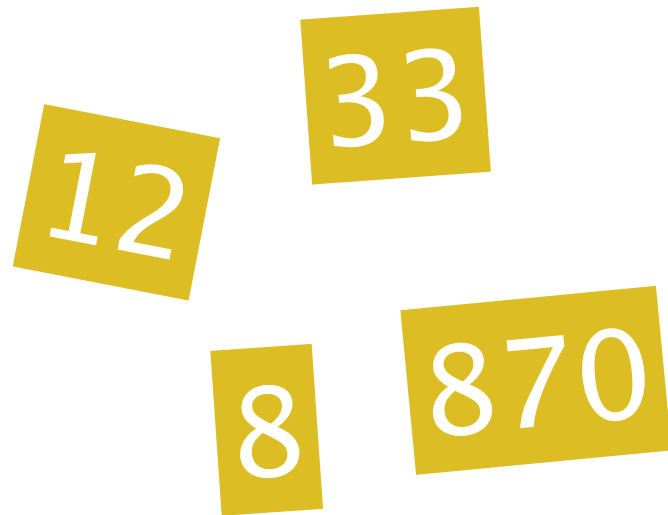


Буквы



# Типы данных

– множество значений и **операций** на этих значениях



Числа

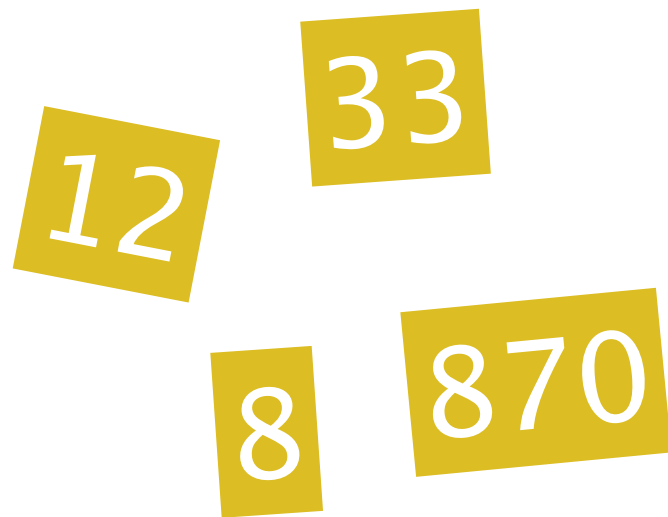


Буквы

$$870 + 12 = 882$$

# Типы данных

– множество значений и **операций** на этих значениях



Числа

$$870 + 12 = 882$$

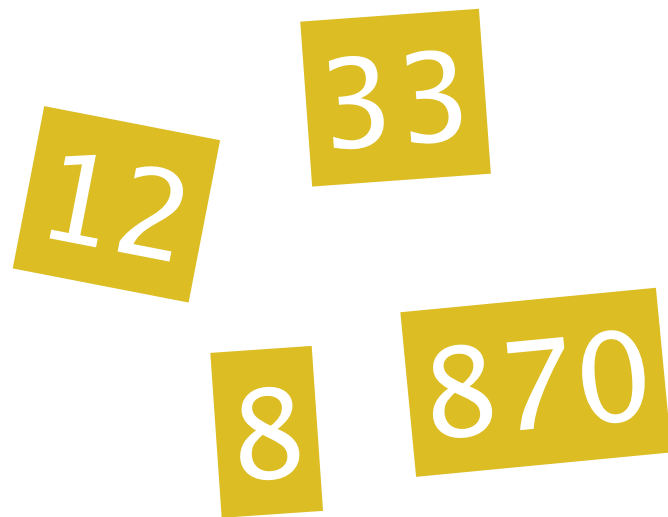


Буквы

$$а + д = ад$$

# Типы данных

– множество значений и **операций** на этих значениях



Числа

$$870 + 12 = 882$$

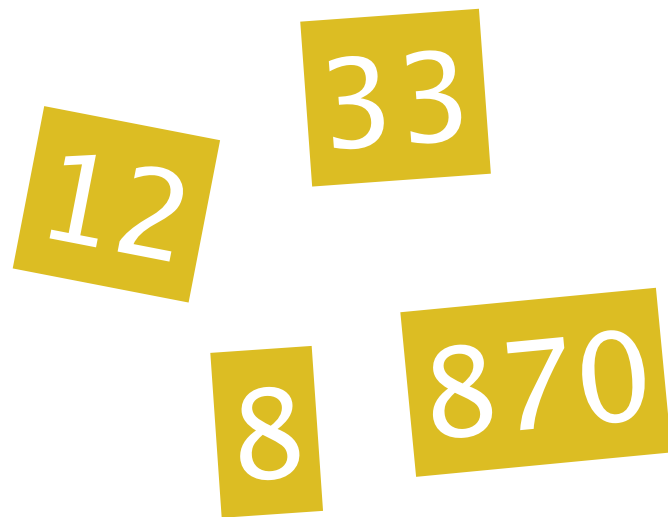


Буквы

$$а + д = ад$$

# Типы данных

– множество значений и **операций** на этих значениях



Числа

$$\begin{array}{ccccc} \boxed{870} & + & \boxed{12} & = & \boxed{882} \\ \text{число} & & \text{число} & & \text{число} \end{array}$$

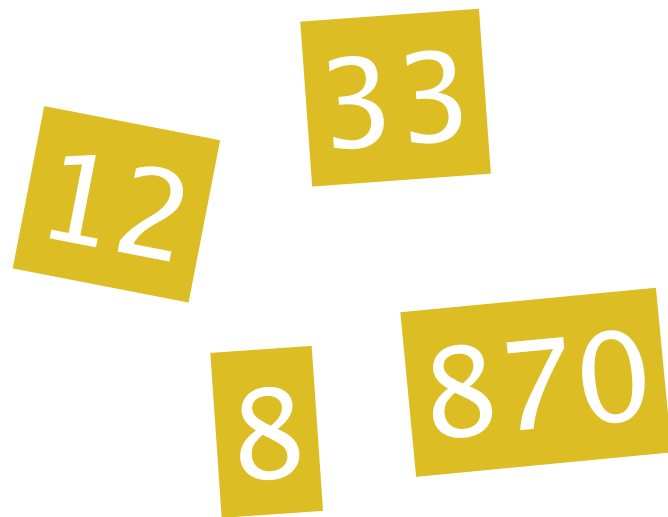


Буквы

$$\boxed{а} + \boxed{д} = \boxed{ад}$$

# Типы данных

– множество значений и **операций** на этих значениях



Числа

$$\begin{array}{ccccc} \boxed{870} & + & \boxed{12} & = & \boxed{882} \\ \text{число} & & \text{число} & & \text{число} \end{array}$$

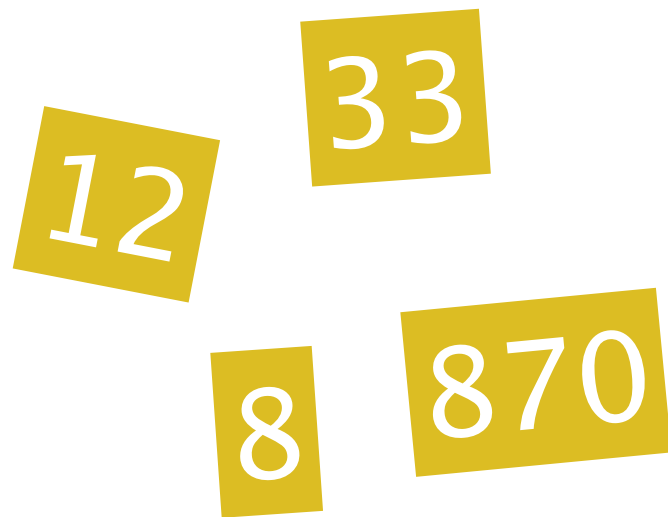


Буквы

$$\begin{array}{ccccc} \boxed{а} & + & \boxed{д} & = & \boxed{ад} \\ \text{буква} & & \text{буква} & & \text{слово} \end{array}$$

# Типы данных

– множество значений и **операций** на этих значениях



Числа

$$\begin{array}{ccccc} \boxed{870} & + & \boxed{12} & = & \boxed{882} \\ \text{число} & & \text{число} & & \text{число} \end{array}$$



Слова

$$\begin{array}{ccccc} \boxed{а} & + & \boxed{д} & = & \boxed{ад} \\ \text{слово} & & \text{слово} & & \text{слово} \end{array}$$

# Простые типы данных

## Целые числа

0: 100

1: 0x3F2A

2: 010

десятичное число

шестнадцатеричное число

восьмеричное число

# Простые типы данных

## Целые числа

0: 100

1: 0x3F2A

2: 010

десятичное число

шестнадцатеричное число

восьмеричное число

## Числа с плавающей запятой

0: 0.2

1: 2e-1

2: 0x10e2 ✗

3: .2

ноль целых, две десятые :)

запись вида  $2 * 10^n$

а вот так уже нельзя

зато можно так



# Простые типы данных

## Числа

```
0: 100
1: 0x3F2A
2: 010
```

```
0: 0.2
1: 2e-1
2: .2
```

## Операции над числами

```
0: 100 + 300 # 400
1: 300 - 100 # 200
2: 3 * 3      # 9
2: 3 ** 3     # 27
2: 6 / 2      # 3.0
2: 6 // 2     # 3
```

# Операции в python

операция

100 + 300

операнд

операнд

# Операции в python

100 + 300

Двухместный оператор

-300

Одноместный оператор

# Операции в python

```
100 + 300
```

Двухместный оператор

```
-300
```

Одноместный оператор

```
300 if 3 + 0 else 100
```

Тернарный (трехместный) оператор

Оператор	Операнд 1	Операнд 2	Результат
+	int	int	int
+	int	float	float
-	int	int	int
-	int	float	float
*	int	int	int
*	int	float	float
/	int	int	float
/	int	float	float
//	int	int	int
//	int	float	float

# Простые типы данных

## Строки

```
0: "Hello"  
1: 'Hello'  
2: """hello"  
3: my friend  
4: lol"""
```

## Операции со строками

```
0: "Hello" + " world!" # "Hello world!"  
1: "Hello" * 2         # "HelloHello"
```

# Простые типы данных

Булевский тип (логический)

```
0: True  
1: False
```

# Простые типы данных

Булевский тип (логический)

```
0: True
1: False
```

Операции с логическими операндами

```
0: True and False      # False
1: True or False       # True
2: not True            # False
3: not (True or False) # False
```



# Простые типы данных

Булевский тип (логический)

```
0: True
1: False
```

Сравнение

```
0: 1 == 2      # False
1: 10 == 10    # True
2: 10 == "10"  # False
```

# Простые типы данных

Специальный тип None

```
0: None
```

# Типизация в Python

- динамическая

# Типизация в Python

- динамическая
- строгая

# Типизация в Python

- динамическая
- строгая

Название	class
Целые числа	int
Числа с плавающей запятой	float
Логический тип	bool
Строки	str
Отсутствие типа	NoneType

# ZOMG TEH TYPE ERROR!!!

```
>>> 0 + '1'
```

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

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

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

# ZOMG TEH TYPE ERROR!!!

```
>>> 0 + '1'
```

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

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

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

# ZOMG TEH TYPE ERROR!!!

```
>>> 0 + '1'
```

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

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

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



# ZOMG TEH TYPE ERROR!!!

```
>>> 0 + '1'
```

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

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

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

# Идентификаторы

# Идентификаторы

- Начинается с любого алфавитного символа Unicode или \_

# Идентификаторы

- Начинается с любого алфавитного символа Unicode или \_

0:	hello	=	0	✓
1:	_hello	=	1	✓
2:	привет	=	2	✓
3:	4ello	=	3	✗

# Идентификаторы

- Начинается с любого алфавитного символа Unicode или \_

0:	hello	=	0	✓
1:	_hello	=	1	✓
2:	привет	=	2	✓
3:	4ello	=	3	✗

- Продолжается любым алфавитным символом, числом и \_

4:	he11o	=	4	✓
5:	привет!	=	5	✗
6:	при-вет	=	6	✗

# Идентификаторы

- Начинается с любого алфавитного символа Unicode или \_

```
0: hello      = 0 ✓  
1: _hello     = 1 ✓  
2: привет    = 2 ✓  
3: 4ello      = 3 ✗
```

- Продолжается любым алфавитным символом, числом и \_

```
4: he1lo      = 4 ✓  
5: привет!    = 5 ✗  
6: при-вет    = 6 ✗
```

- Регистр имеет значение

```
7: Hello      = 7 ✓  
8: hello      = 8 ✓
```

# Идентификаторы

Переменные в программе

```
>>> a = 10  
>>> b = 20  
>>> a + b  
30  
>>>
```

# Идентификаторы

## Переменные в программе

```
>>> a = 10
```

```
>>> b = 20
```

```
>>> a + b
```

```
30
```

```
>>>
```

```
>>> a = b
```

```
>>> a
```

```
20
```



# Идентификаторы

## Переменные в программе

```
>>> a = 10
>>> b = 20
>>> a + b
30
>>>
>>> a = b
>>> a
20
>>> a * 2
400
```

# БЛОКИ КОДА

## Code suites

# БЛОКИ КОДА

## Code suites

```
01: for question in questions:
02:     text = question.get('text', '')
03:     col = int(question.get('col'))
04:
05:     cols += [col];
06:
07:     while True:
08:         try:
09:             ans = raw_input(text + '\n')
10:             result = handle(ans)
11:             break
12:         except ValueError:
13:             print 'Incorrect value. \n'
```

# БЛОКИ КОДА

## Code suites

```
01: for question in questions:
02:     ↔ text = question.get('text', '')
03:     col = int(question.get('col'))
04:
05:     cols += [col];
06:
07:     while True:
08:         ↔ try:
09:         ↔ ans = raw_input(text + '\n')
10:             result = handle(ans)
11:             break
12:         except ValueError:
13:             print 'Incorrect value. \n'
```

# БЛОКИ КОДА

## Code suites

```
01: for question in questions:
02:     ↔ text = question.get('text', '')
03:     4 sp. col = int(question.get('col'))
04:
05:     cols += [col];
06:
07:     while True:
08:     8 sp. ↔ try:
09:     12 sp. ↔ ans = raw_input(text + '\n')
10:             result = handle(ans)
11:             break
12:     except ValueError:
13:         print 'Incorrect value. \n'
```

```
1  for question in questions:
2      text = question.get('text', '')
3      col = int(question.get('col'))
4
5      cols += [col];
6
7      while True:
8          try:
9              ans = raw_input(text + '\n')
10             result = handle(ans)
11             break
12         except ValueError:
13             print 'Incorrect value. \n'
```

```
1  for question in questions:
2  text = question.get('text', '')
3      col = int(question.get('col'))
4
5      cols += [col];
6
7      while True:
8          try:
9              ans = raw_input(text + '\n')
10             result = handle(ans)
11             break
12         except ValueError:
13             print 'Incorrect value. \n'
```

```
1  for question in questions:
2  text = question.get('text', '')
3      col = int(question.get('col'))
4
5      cols += [col];
6
7      while True:
8          try:
9              ans = raw_input(text + '\n')
10             result = handle(ans)
11             break
12         except ValueError:
13             print 'Incorrect value. \n'
```



# Пару слов о Code Style

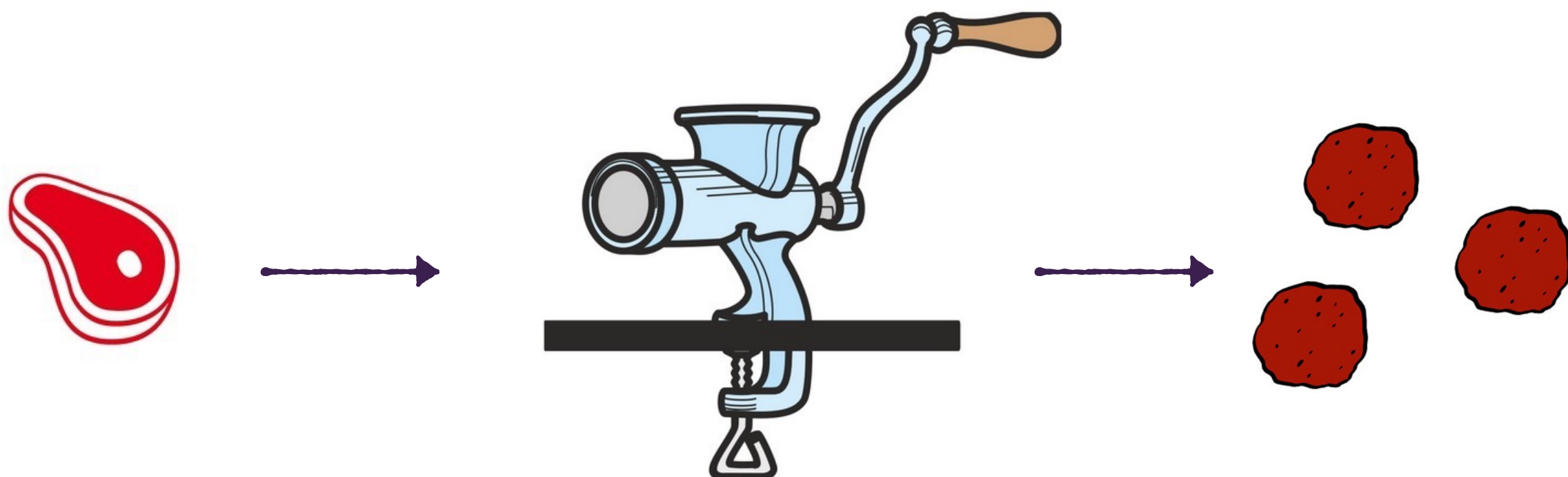
# Пару слов о Code Style

[PEP 8 - руководство по написанию кода на Python](#)

# Функции

$$f(x) : X \rightarrow Y$$

# Функции

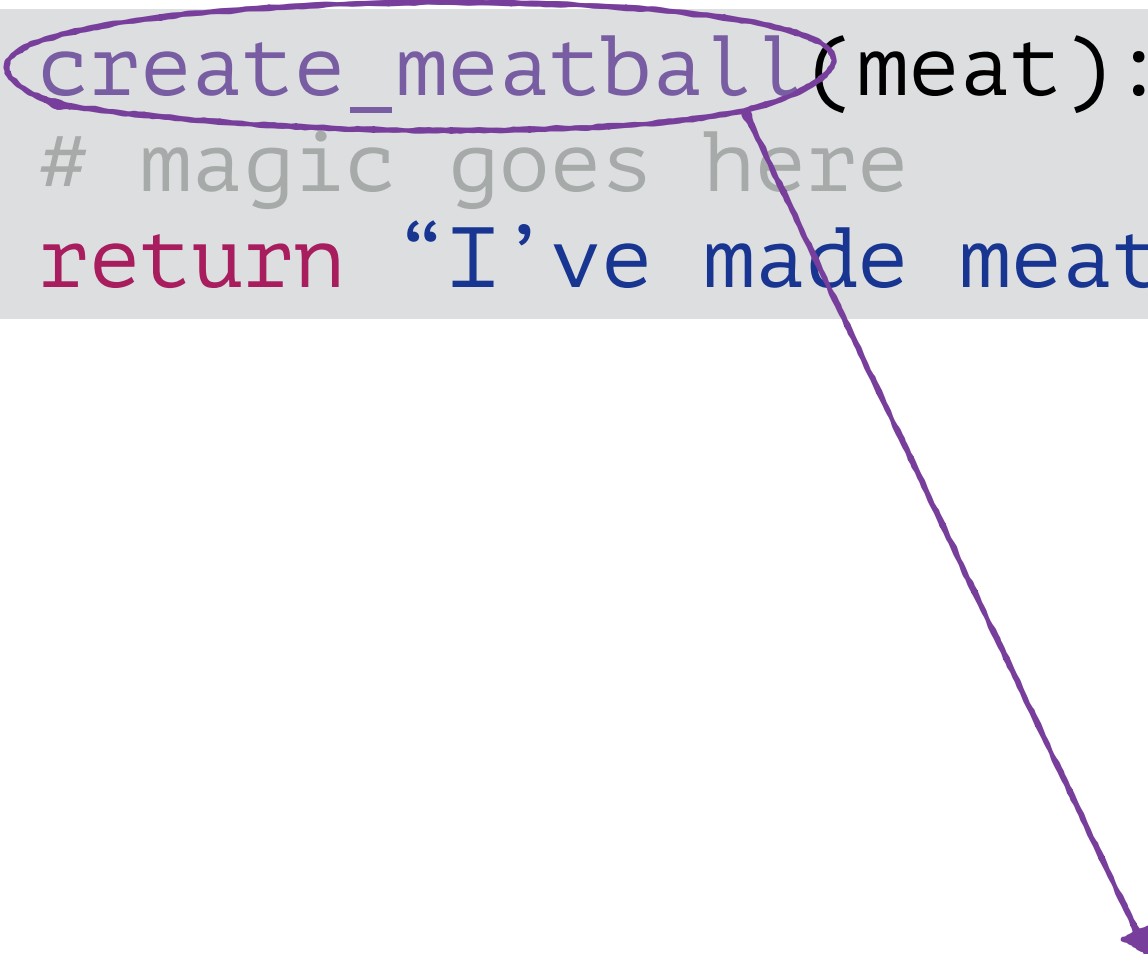


# Функции

```
0: def create_meatball(meat):  
1:     # magic goes here  
2:     return "I've made meatball from " + meat
```

# Функции

```
0: def create_meatball(meat):  
1:     # magic goes here  
2:     return "I've made meatball from " + meat
```



**Название функции**

# Функции

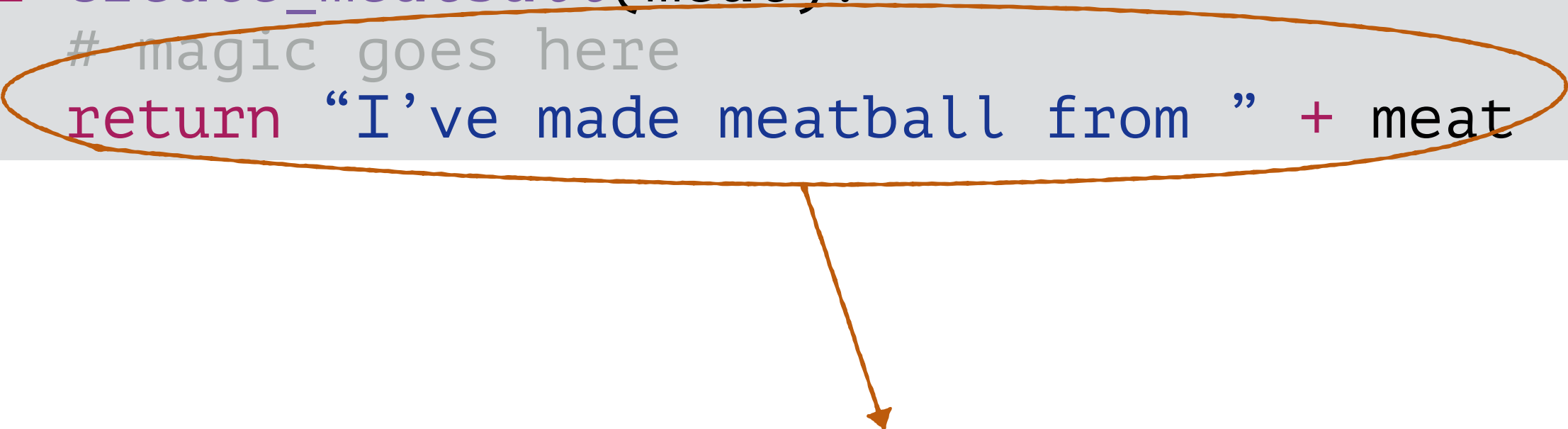
```
0: def create_meatball(meat):  
1:     # magic goes here  
2:     return "I've made meatball from " + meat
```

**Аргументы функции**



# Функции

```
0: def create_meatball(meat):  
1:     # magic goes here  
2:     return "I've made meatball from " + meat
```



**Тело функции**



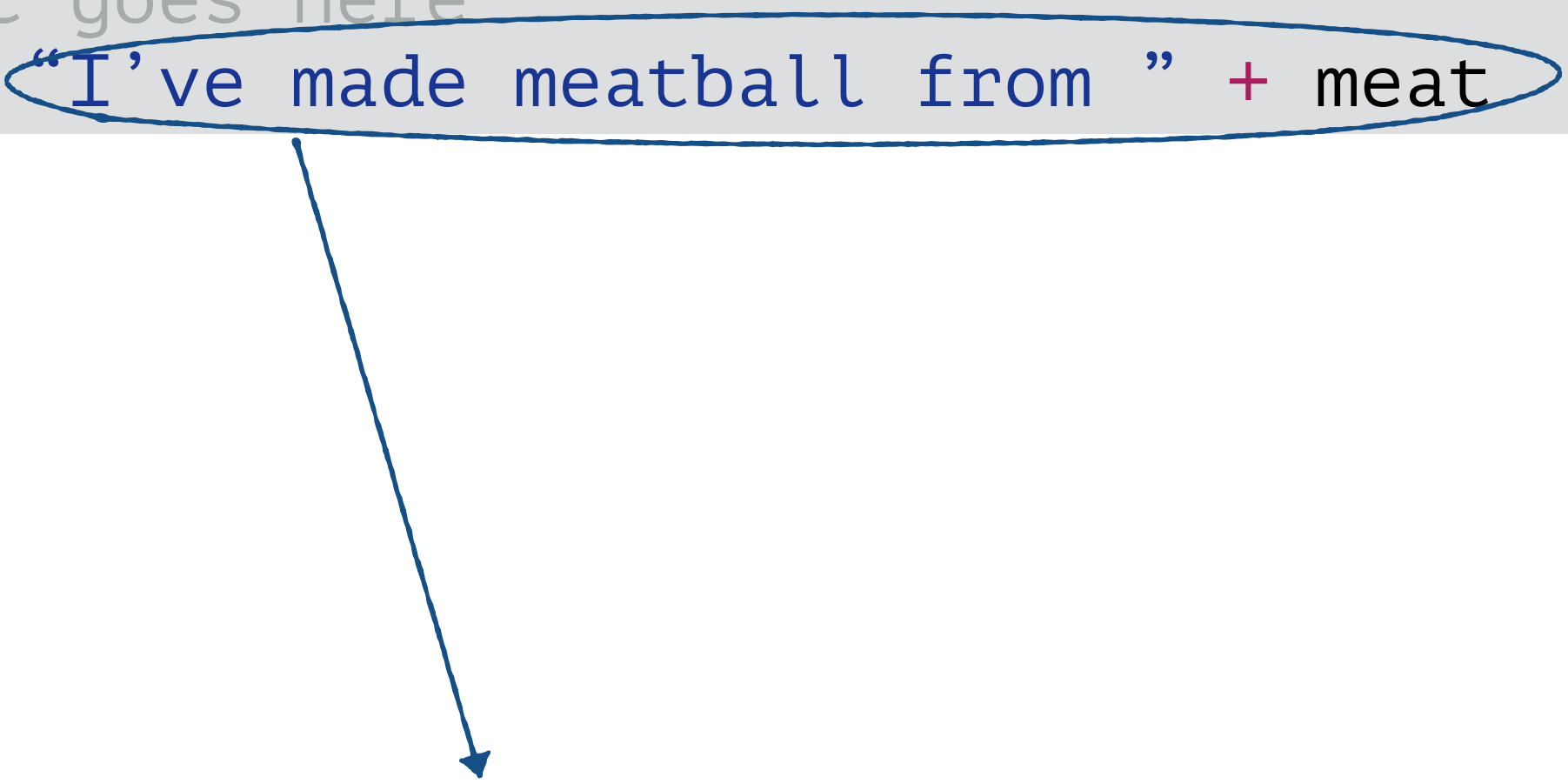
# Функции

```
0: def create_meatball(meat):  
1:     # magic goes here  
2:     return "I've made meatball from " + meat
```

  
Отступ

# Функции

```
0: def create_meatball(meat):  
1:     # magic goes here  
2:     return "I've made meatball from " + meat
```



**Возвращаемое значение**

# Функции

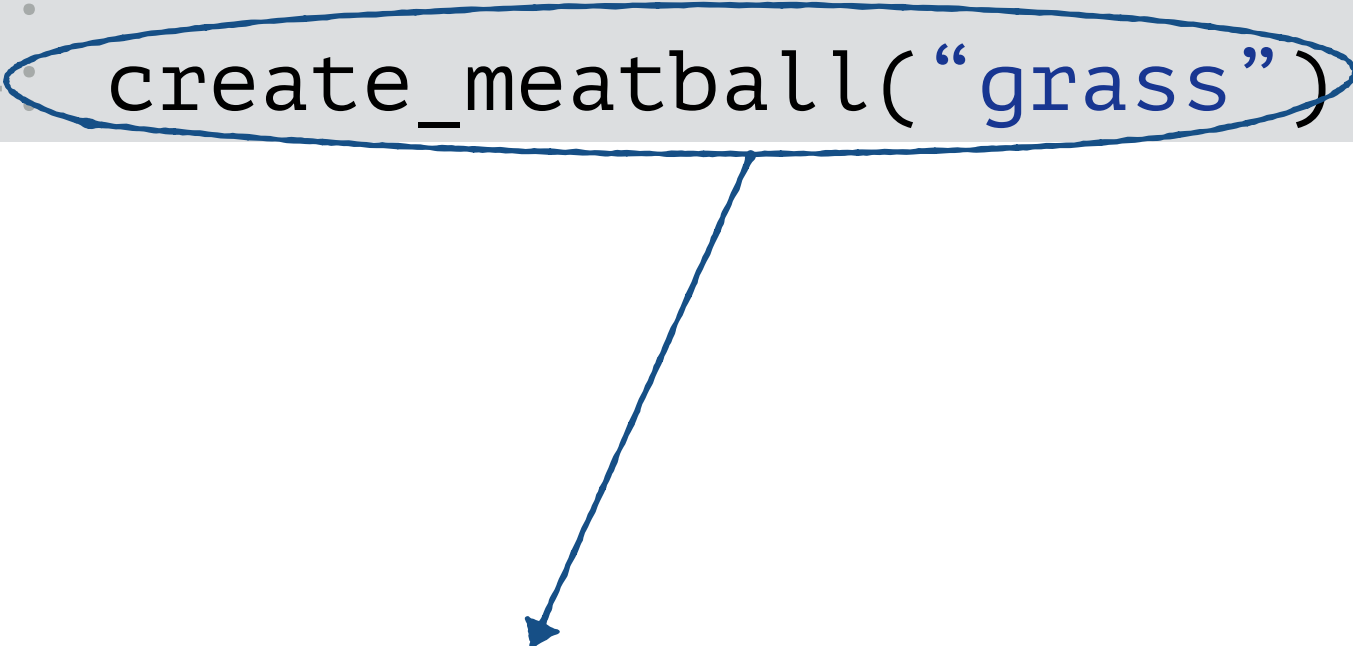


# Функции

```
0: def create_meatball(meat):  
1:     # magic goes here  
2:     return "I've made meatball from " + meat  
3:  
4: create_meatball("grass")
```

# Функции

```
0: def create_meatball(meat):  
1:     # magic goes here  
2:     return "I've made meatball from " + meat  
3:  
4: create_meatball("grass")
```



**Вызов функции (call)**

# Функции

```
0: def create_meatball(meat):  
1:     # magic goes here  
2:     return "I've made meatball from " + meat  
3:  
4: create_meatball("grass")
```



"I've made meatball from grass"

# Функции

Анонимные функции (лямбда-функции)

```
0: def add(x, y):  
1:     return x + y
```

# Функции

Анонимные функции (лямбда-функции)

```
0: def add(x, y):  
1:     return x + y
```

```
0: add = lambda x, y: x + y
```



# Функции

Анонимные функции (лямбда-функции)

```
0: def add(x, y):  
1:     return x + y
```

```
0: add = lambda x, y: x + y
```

```
0: add(2, 3) # -> 5
```

# Функции

– это объекты первого класса

```
0: def add(x, y):  
1:     return x + y
```

```
0: another_add = add
```

```
0: another_add(2, 3) # -> 5
```

# Функции

## Параметры по-умолчанию

```
01: def add(x, y):  
02:     if not x:  
03:         x = 0  
05:     return x + y
```

# Функции

## Параметры по-умолчанию

```
01: def add(x, y):  
02:     if not x:  
03:         x = 0  
05:     return x + y
```

```
01: add(20) # 20
```

# Функции

## Параметры по-умолчанию

```
01: def add(x, y=0):  
02:     return x + y
```

```
01: add(20) # 20
```

# Функции

## Параметры по-умолчанию

```
01: def add(x, y=0):  
02:     return x + y
```

```
01: add(20) # 20
```

# Функции

## Строки документации

```
01: def add(x, y=0):  
02:     "Returns sum of given two numbers"  
03:  
04:     return x + y
```

# Функции

Строки документации

```
01: def add(x, y=0):  
02:     "Returns sum of given two numbers"  
03:  
04:     return x + y
```

```
$ pydoc add
```



# ФУНКЦИИ

Строки документации

```
Help on module add:
```

```
NAME
```

```
    add
```

```
FILE
```

```
    /Users/zeffirsky/Documents/python/add.py
```

```
FUNCTIONS
```

```
    add(x, y=0)
```

```
        Returns sum of given two numbers
```

# Функции

Строки документации

```
01: def add(x, y=0):  
02:     "Returns sum of given two numbers"  
03:  
04:     return x + y
```

```
>>> add.__doc__
```