

Лекция 1

Введение.

Основные функции.

История

- Назван в честь сериала “Monty python's flying circus”
- Python 3.0 вышел без обратной совместимости с предыдущими версиями, поэтому версия 2.x до сих пор популярна



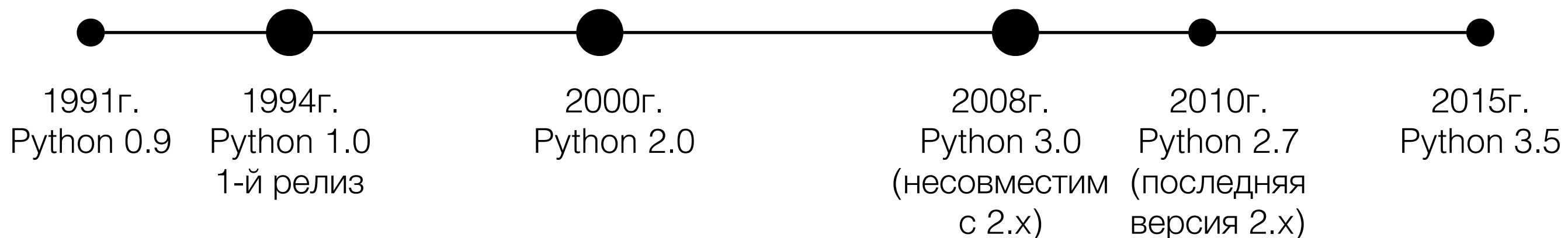
Автор: Гвидо ван Россум.
Benevolent Dictator for Life.

История

- Назван в честь сериала “Monty python's flying circus”
- Python 3.0 вышел без обратной совместимости с предыдущими версиями, поэтому версия 2.x до сих пор популярна
- Используется, как скриптовый язык в Yandex, Google и т.д.
- Python сейчас — стандарт де-факто для машинного обучения, обработки текстов



Автор: Гвидо ван Россум.
Benevolent Dictator for Life.



Чем хорош python

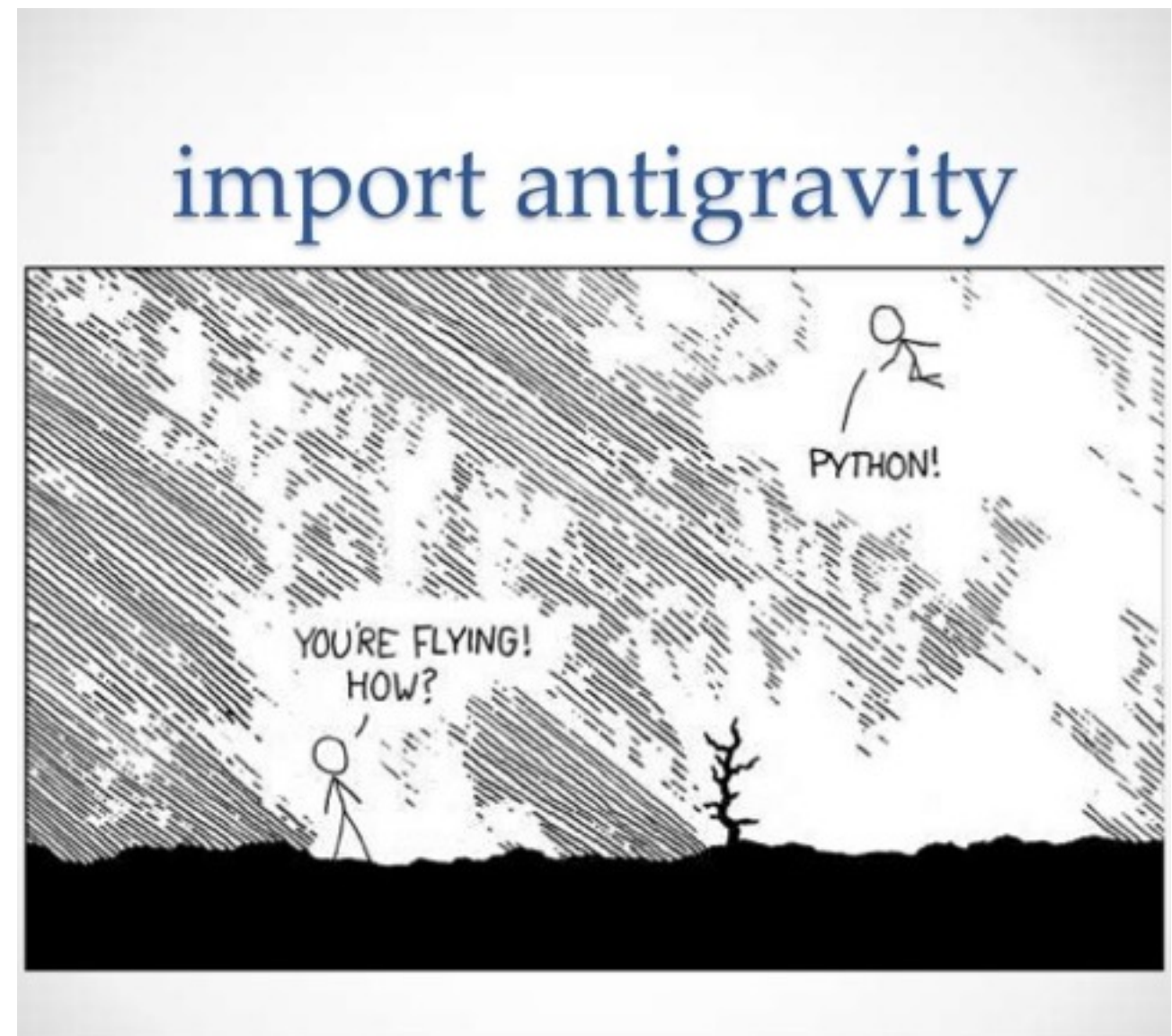
Высокая скорость разработки

- *Объем кода* значительно меньше по сравнению с C++, C#, Java.
- *Легко читать* и писать: мало непонятных символов, есть отступы
- Богатая *стандартная библиотека* (в том числе, для обработки текста)
- Крупное и *активное сообщество*: много сторонних библиотек, которые хорошо поддерживаются

Чем хорош python

Высокая скорость разработки

- *Объем кода* значительно меньше по сравнению с C++, C#, Java.
- *Легко читать* и писать: мало непонятных символов, есть отступы
- Богатая *стандартная библиотека* (в том числе, для обработки текста)
- Крупное и *активное сообщество*: много сторонних библиотек, которые хорошо поддерживаются



Чем плох python

Низкая скорость исполнения.

- В чистом виде *медленнее C в 10-1000 раз.*
- Проблемы с реализацией многопоточности.

Чем плох python

Низкая скорость исполнения.

- В чистом виде *медленнее C в 10-1000 раз.*
- Проблемы с реализацией многопоточности.

Решение:

- Использовать в связке с другим языком программирования: медленная часть — на быстром языке (C, C++), сложная часть — на простом языке (Python).
- Библиотеки, ускоряющие вычисления (numru), cython-расширения и т.д.
- Если скорость работы первостепенна — использовать другой язык.

Философия

Simple is better than complex.

Complex is better than complicated.

Readability counts.

There should be one — and preferably only one — obvious way to do it.



Интерактивный режим

Python — *интерпретируемый* язык, программу не нужно компилировать перед запуском.

Есть интерактивный или *консольный режим* — программа выполняется сразу после набора команды.

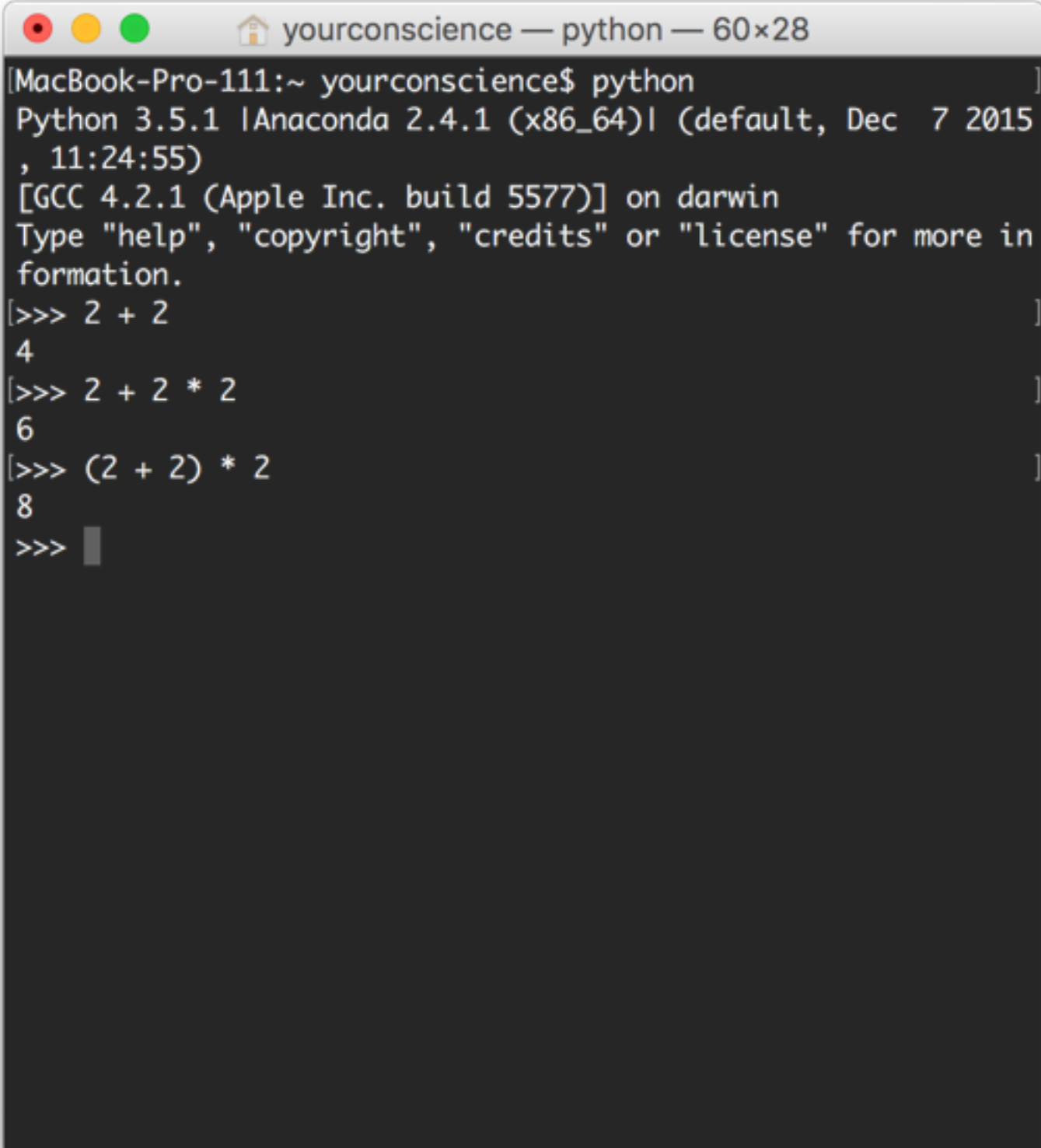
Удобно использовать в качестве *калькулятора* или для проверки работы различных функций.

Интерактивный режим

Python — *интерпретируемый* язык, программу не нужно компилировать перед запуском.

Есть интерактивный или *консольный режим* — программа выполняется сразу после набора команды.

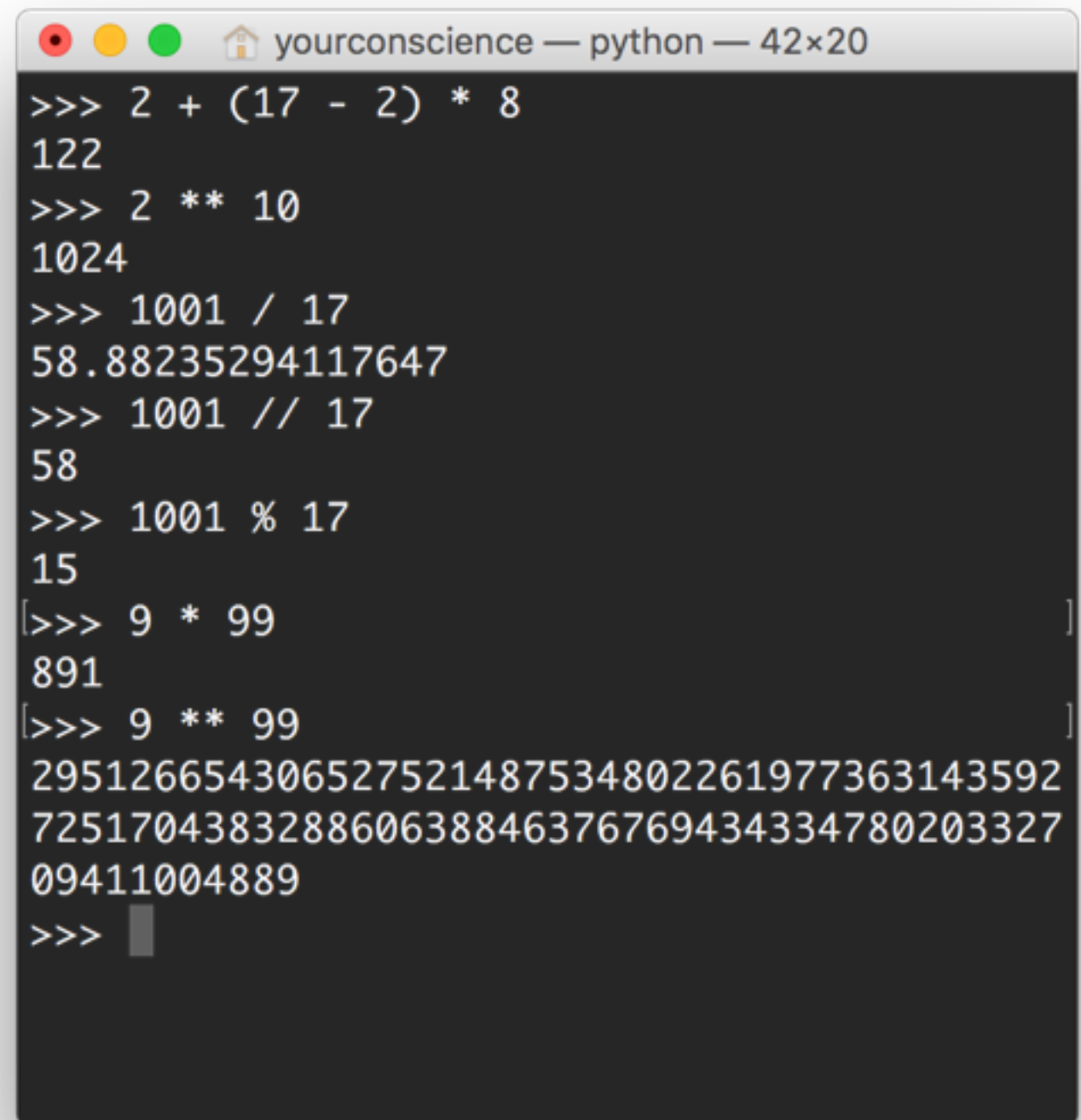
Удобно использовать в качестве *калькулятора* или для проверки работы различных функций.

A screenshot of a macOS terminal window titled "yourconscience — python — 60x28". The terminal shows the execution of the Python interpreter. The prompt is "[MacBook-Pro-111:~ yourconscience\$ python]". The output shows the Python version "Python 3.5.1 |Anaconda 2.4.1 (x86_64)| (default, Dec 7 2015, 11:24:55)", the compiler "[GCC 4.2.1 (Apple Inc. build 5577)] on darwin", and instructions to type "help", "copyright", "credits" or "license" for more information. The interactive prompt ">>>" is shown three times with the following inputs and outputs: ">>> 2 + 2" results in "4", ">>> 2 + 2 * 2" results in "6", and ">>> (2 + 2) * 2" results in "8". The prompt ">>>" is shown again with a cursor, indicating the shell is ready for further input.

```
[MacBook-Pro-111:~ yourconscience$ python
Python 3.5.1 |Anaconda 2.4.1 (x86_64)| (default, Dec 7 2015, 11:24:55)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more in
formation.
>>> 2 + 2
4
>>> 2 + 2 * 2
6
>>> (2 + 2) * 2
8
>>> 
```

Ввод, вывод и основные конструкции

Арифметические операции



```
yourconscience — python — 42x20
>>> 2 + (17 - 2) * 8
122
>>> 2 ** 10
1024
>>> 1001 / 17
58.88235294117647
>>> 1001 // 17
58
>>> 1001 % 17
15
>>> 9 * 99
891
>>> 9 ** 99
295126654306527521487534802261977363143592
725170438328860638846376769434334780203327
09411004889
>>>
```

Арифметические операции

Сложение, умножение, вычитание — обычные.

Два вида деления — точное и целочисленное, с отбрасыванием остатка.

Есть операции взятия по модулю и степень.

Длинная арифметика.

```
yourconscience — python — 42x20
>>> 2 + (17 - 2) * 8
122
>>> 2 ** 10
1024
>>> 1001 / 17
58.88235294117647
>>> 1001 // 17
58
>>> 1001 % 17
15
>>> 9 * 99
891
>>> 9 ** 99
295126654306527521487534802261977363143592
725170438328860638846376769434334780203327
09411004889
>>>
```

Переменные и типы

```
yourconscience — python — 42x20
>>> x = 1
>>> y = 2
>>> x + y
3
>>> z = 'apple'
>>> z += 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to s
tr implicitly
>>> int('2')
2
>>> str(2)
'2'
>>> float(2)
2.0
>>>
>>>
>>>
>>>
```

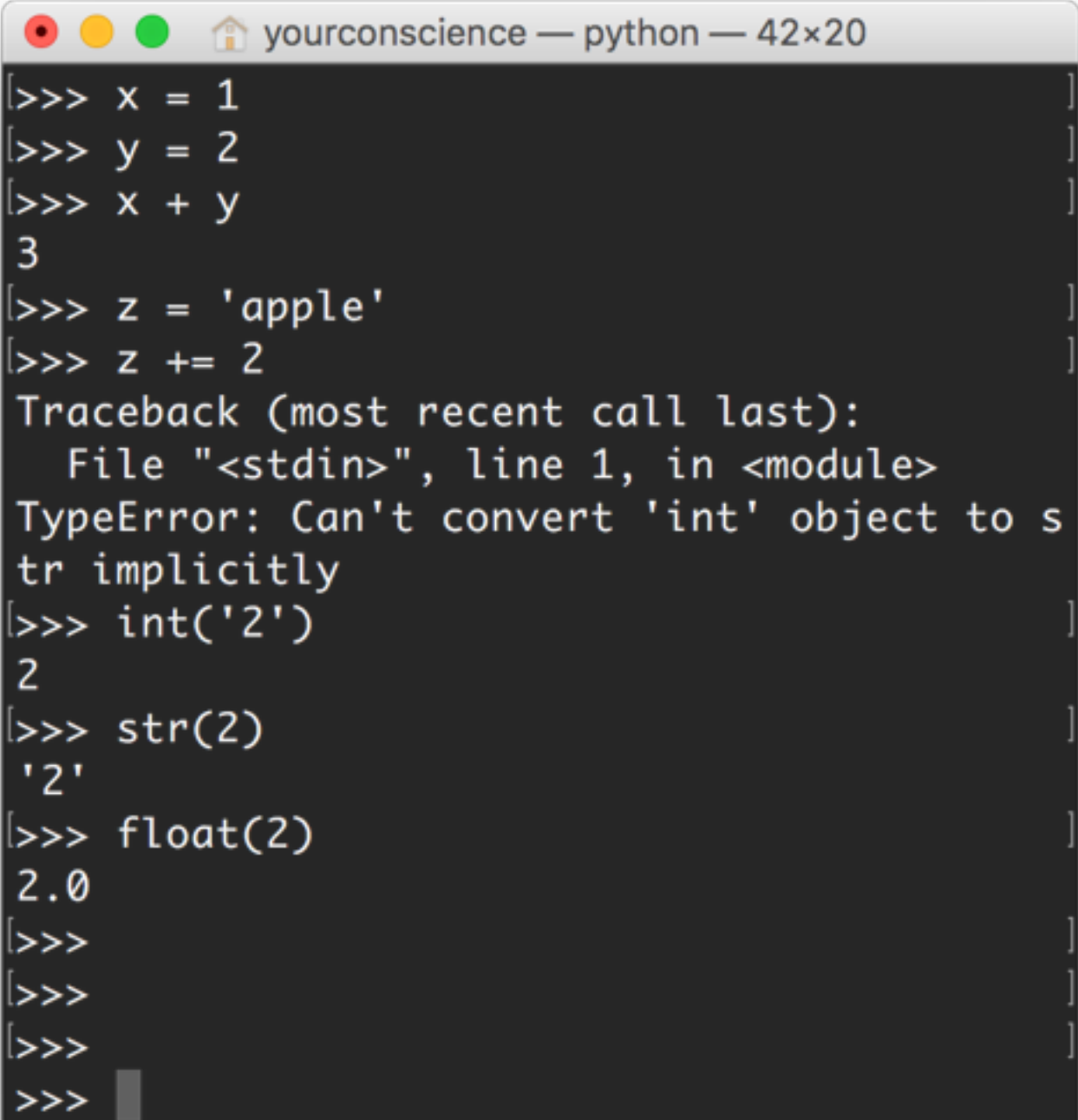
Переменные и типы

Объявление переменной происходит одновременно с инициализацией — нужно просто задать ее значение.

В питоне *динамическая типизация*. Это значит, что переменные имеют типы, но явно задавать их не нужно.

В питоне есть неявное приведение типов, но можно и явно *конвертировать типы*.

Базовые типы: int, float, str, bool.



```
yourconscience — python — 42x20
>>> x = 1
>>> y = 2
>>> x + y
3
>>> z = 'apple'
>>> z += 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to s
tr implicitly
>>> int('2')
2
>>> str(2)
'2'
>>> float(2)
2.0
>>>
>>>
>>>
>>>
```


Ввод и вывод

[illegible]

Логические операции

```
yourconscience — python — 42x20
>>> 1 > 0
True
>>> 1 > 0 and 0 > 1
False
>>> 1 > 0 or 0 > 1
True
>>> False or True and False
False
>>> not 0
True
>>> 1 or False and ''
1
>>>
>>>
>>>
>>>
>>>
>>>
>>>
```

Логические операции

False и True пишутся с большой буквы — зарезервированные слова.

and приоритетнее, чем *or*.

Можно сравнивать разные типы.

X and Y or Z — аналог
if X then Y else Z

```
yourconscience — python — 42x20
>>> 1 > 0
True
>>> 1 > 0 and 0 > 1
False
>>> 1 > 0 or 0 > 1
True
>>> False or True and False
False
>>> not 0
True
>>> 1 or False and ''
1
>>>
>>>
>>>
>>>
>>>
>>>
>>>
```

Условный оператор

```
print('Please, tell me your  
birthday date:')  
day = int(input())  
correct = day > 0 and day <= 31  
if not correct:  
    print('Looks like you  
misspelled.')  
else:  
    print('Thank you!')
```

Условный оператор

Логические блоки выделяются *отступами*. Это аналог фигурных скобок в C/C++, *begin/end* в Паскале. Размер блока может быть любым, содержать другие блоки и т.д.

Перед началом логического блока ставится *двоеточие*.

Условный оператор имеет так же краткую форму: *A if cond else B*. В этой форме *else* обязателен.

```
print('Please, tell me your  
birthday date:')  
day = int(input())  
correct = day > 0 and day <= 31  
if not correct:  
    print('Looks like you  
misspelled.')
```

```
else:  
    print('Thank you!')
```

Условный оператор

```
print('Please, tell me the day of your birth:')
day = int(input())
correct = day > 0 and day <= 31
if not correct:
    print('Looks like you misspelled.')
else:
    print('Thank you!')
    if day == 11:
        print('Oh, just like my birthday!')
```

```
Please, tell me the day of your birth:
>>11
Thank you!
Oh, just like my birthday!
```


Цикл while

```
i = 1
s = 0
while i <= 10:
    s += i
    i += 1

sudo = 'admin'
while input('Enter your login: ') != sudo:
    print('Pffff, you have no permissions, man')
print('Hello, Mr. admin, what can I do for you?')
request = input()
```

Задачи

1. Посчитать в консоли сумму чисел от 1 до 10 , возвести ее в квадрат и вывести ответ по модулю 7 .
2. Написать программу, которая запрашивает у пользователя число и строит по нему *сиракузскую последовательность*, выводя по одному числу в строке.

Последовательность строится так:

- Если n делится на 2 , то $n \rightarrow n / 2$
- Иначе $n \rightarrow 3n + 1$

[О сиракузской последовательности на вики](#)