Программирование Python

Философия Python

Разработчики языка Python придерживаются определённой философии программирования, называемой «The Zen of Python» («Дзен Питона», или «Дзен Пайтона»). Её текст выдаётся интерпретатором Python по команде import this (работает один раз за сессию).

В целом она подходит к программированию на любом языке.

- Текст философии
- Красивое лучше, чем уродливое.
- Явное лучше, чем неявное.
- Простое лучше, чем сложное.
- Сложное лучше, чем запутанное.
- Плоское лучше, чем вложенное.
- Разреженное лучше, чем плотное.
- Читаемость имеет значение.
- Особые случаи не настолько особые, чтобы нарушать правила.
- При этом практичность важнее безупречности.
- Ошибки никогда не должны замалчиваться.
- Если они не замалчиваются явно.
- Встретив двусмысленность, отбрось искушение угадать.
- Должен существовать один и, желательно, только один очевидный способ сделать это.
- Хотя он поначалу может быть и не очевиден, если вы не голландец.
- Сейчас лучше, чем никогда.
- Хотя никогда зачастую лучше, чем прямо сейчас.
- Если реализацию сложно объяснить — идея плоха.
- Если реализацию легко объяснить — идея, возможно, хороша.
- Пространства имён — отличная штука! Будем делать их больше!

In [ ]:

```
import this
```

Python как калькулятор

In [ ]:

```
4*7
```

In [ ]:

```
3*(2+5)
```

In [ ]:

```
5^3 #что получится
```

In [ ]:

```
5**3
```

Со сложением, умножением и степенью всё должно быть понятно. Деление, на первый взгляд, тоже не предвещает проблем:

In [ ]:

```
12/3
```

Внимание! Поведение Python версии 2 было другим: он всегда делил нацело (отбрасывая остаток). Это приводило к огромному количеству проблем, когда разработчик ожидал, что деление будет «обычное», а получал целочисленное. В Python 3 деление происходит в числах с плавающей точкой; если вам нужно целочисленное, вы можете использовать двойной слеш (`//`).

In [ ]:

```
3//2 # что получится?
```

Как обстоят дела с другими операциями? Попробуем извлечь квадратный корень:

In [ ]:

```
sqrt(4)
```

Соответствующая функция есть в стандартном модуле `math` . Чтобы ей воспользоваться, нужно импортировать этот модуль. Это можно сделать разными способами.

In [ ]:

```
import math
math.sqrt(4)
```

После того, как модуль math импортирован, вы можете узнать, какие ещё в нём есть функции. В IPython Notebook для этого достаточно ввести имя модуля, поставить точку и нажать кнопку «Tab». Вот, например, синус:

In [ ]:

```
math.sin(0)
```

Приведенный синтаксис может оказаться неудобным, если вам часто приходится вызывать какие-то математические функции. Чтобы не писать каждый раз слово «math», можно импортировать из модуля конкретные функции.

In [ ]:

```
from math import sqrt
```

In [ ]:

```
sqrt(4)
```

Вещественные числа и погрешности

Вещественные числа в программировании не так просты. Вот, например, посчитаем синус числа π:

In [ ]:

```
from math import pi, sin
sin(pi)
# получится 0?
```

Непонятный ответ? Во-первых, это так называемая [компьютерная форма экспоненциальной записи (https://ru.wikipedia.org/wiki/%D0%AD%D0%BA%D1%81%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D](https://ru.wikipedia.org/wiki/%D0%AD%D0%BA%D1%81%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D0%B0%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D0%B0_%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D0%B8_%D1%87%D0%B8%D1%81%D0%B5%D0%BB) чисел. Она удобна, если нужно уметь записывать очень большие или очень маленькие числа: `1.2E2` означает $1{,}2 \cdot 10^2$, то есть 1200, а `2.4e-3` — то же самое, что $2{,}4 \cdot 10^{-3} = 0{,}0024$. Результат, посчитанный Python для $\sin \pi$, имеет порядок 10^{-16} — это очень маленькое число, близкое к нулю. Почему не «настоящий» ноль? Все вычисления в вещественных числах делаются компьютером с некоторой ограниченной точностью, поэтому зачастую вместо «честных» ответов получаются такие приближенные. К этому надо быть готовым.

Переменные

In [ ]:

```
x = 4
x
```

В Питоне не нужно как-то специально объявлять переменную — достаточно просто присвоить ей значение, и можно пользоваться!

In [ ]:

```
x + 2
```

In [ ]:

```
x = x + 2
```

In [ ]:

```
x
```

Переменные могут хранить не только числа, но и другие данные. Например, строки. Мы столкнёмся с этим прямо в следующем разделе.

Ввод данных

Работа в Jupyter / IPython Notebook редко требует писать код, который сам по себе запрашивает данные с клавиатуры, но для других приложений (и в частности для домашних работ) это может потребоваться. К тому же, написание интерактивных приложений само по себе забавное занятие. Напишем, например, программу, которая здоровается с нами по имени.

In [ ]:

```
name = input("Введите ваше имя: ")
print("Привет,",name)
```

In [ ]:

```
name
```

Попробуем теперь написать программу «удвоитель». Она должна будет принимать на вход число, удваивать его и возвращать результат.

In [ ]:

```
x = input("Введите какое-нибудь число: ")
print(x*2)
```

Что-то здесь не то. Конечно, в некотором смысле компьютер сделал то, что мы его просили — было одно число 12, а стало два, но мы-то хотели чего-то другого. Почему так получилось? Дело в том, что `input` всегда возвращает строку (он же не знает, что мы введём — цифры или какие-то другие символы; он не понимает, что в запросе сказан «Введите какое-нибудь число» — для него это просто какая-то строка). И в переменной `x` сейчас не число `12`, а строка `'12'` (обратите внимание на апострофы).

In [ ]:

```
x
```

Чтобы работать с ним как с числом, надо его превратить из строкового типа в числовой. Например, если мы хотим работать только с целыми числами, то подходящим будет тип `int` (от слова `integer`, целое число).

In [ ]:

```
int(x)
```

Видите, апострофов нет — это уже число. Слово `int` это одновременно и обозначение типа данных (целые числа) и функция, которая превращает то, что ей передали в качестве аргумента, в указанный тип данных.

Перепишем программу-удвоитель следующим образом:

In [ ]:

```
x_str = input("Введите какое-нибудь число: ")
x = int(x_str)
print(x*2)
```

Совсем другое дело! Теперь в `x_str` лежит строчка, которую мы ввели, в `x` помещается результат её преобразования в целое число, и дальше это число удваивается и выводится на экран. Можно было бы не вводить дополнительную переменную (и сэкономить строчку):

In [ ]:

```
x = int(input("Введите какое-нибудь число: "))
print(x*2)
```

Попробуем ещё раз.

In [ ]:

```
x = int(input("Введите какое-нибудь число: "))
print(x*2)
```

Я попробовал ввести число `12.34` (нецелое) и Python не смог привести его в типу `int` (то есть сделать из строчки `"12.34"` целое число и выдал ошибку. Если бы мы хотели работать с нецелыми числами, нам нужно было использовать тип `float`.

In [ ]:

```
x = float(input("Введите какое-нибудь число (не обязательно целое): "))
print(x*2)
```

Всё работает!

> Кстати, чтобы превратить число в строчку можно использовать функцию `str`. Вообще, название любого типа данных, будучи вызванным как функция, пытается преобразовать свой аргумент в соответствующий тип данных.

Практичекое задание: числа Фибоначчи

Числа Фибоначчи или *последовательность Фибоначчи* — последовательность чисел, начинающаяся с двух единиц, и такая, что очередное число в этой последовательности равно сумме двух предыдущих. Формально можно определить её следующим образом:

$a_1 = 1;$

$a_2 = 1;$

$a_{n+1} = a_n + a_{n-1}$ для всех $n > 2$.

Например, $a_3 = 1 + 1 = 2$, $a_4 = 2 + 1 = 3$.

Задача: посчитать 15-е число Фибоначчи

In [ ]:

```
a = 1 # первое число
b = 1 # второе число
i = 2 # номер того числа, которое находится в переменной b (сейчас это a_2)
```

Мы будем хранить два последних числа, поскольку именно они определяют следующее, а также номер последнего найденного числа (переменная i).

In [ ]:

```
c = a + b # нашли следующее число
i = i + 1 # увеличили i на 1
a = b   # значение a нам уже не нужно, а вот значение b ещё пригодится
b = c   # запомнили вычисленное значение
print(i, b)
```

Питон выполняет команды последовательно, строчку за строчкой, поэтому порядок следоавния команд очень важен. Выполняя эту ячейку несколько раз, вы будете получать каждый раз очередное число Фибоначчи.

Контрольный вопрос. Что произойдёт, если мы поменяем местами две строчки до `print` ?

In [ ]:

```
# Повторим этот код ещё раз

c = a + b # нашли следующее число
i = i + 1 # увеличили i на 1
a = b   # значение a нам уже не нужно, а вот значение b ещё пригодится
b = c   # запомнили вычисленное значение
print(i, b)
```

In [ ]:

```
# И ещё раз

c = a + b # нашли следующее число
i = i + 1 # увеличили i на 1
a = b   # значение a нам уже не нужно, а вот значение b ещё пригодится
b = c   # запомнили вычисленное значение
print(i, b)
```

In [ ]:

```
# И ещё раз

c = a + b # нашли следующее число
i = i + 1 # увеличили i на 1
a = b   # значение a нам уже не нужно, а вот значение b ещё пригодится
b = c   # запомнили вычисленное значение
print(i, b)
```

Совет. Можно было не копироать эту ячейку с кодом много раз, а просто запускать одну и ту же ячейку неколько раз. Проще всего это сделать с помощью комбинации *Ctrl+Enter*. И, конечно, на практике мы не будем вручную выполнять один код много раз — для этого есть *циклы*. Но с ними мы познакомимся на следующей лекции.