Введение в Python

Яцулевич Владимир Владимирович

1. ВВЕДЕНИЕ

Python — один из самых популярных языков программирования в настоящее время. Он используется практически везде: веб-разработка, геймдев, машинное обучение, научные исследования и много чего другого.

Начнём изучения с простой команды.

```
print('Hello world!')

Out[]: Hello World!
```

Данная команда выводит строку `Hello world!` на экран. Чуть позже опишем более детально, что означает каждый символ в данной команде.

Изучение программирования стоит начать с изучения двух фундаментальных понятий: переменная и функция. Что умеет делать компьютер? В основном компьютер умеет хранить информацию и её обрабатывать. Как раз для хранения информации используются переменные, а для обработки — функции. Представьте себе коробку, в которую вы положили какой-то предмет. А на этой коробке вы сделали отметку.

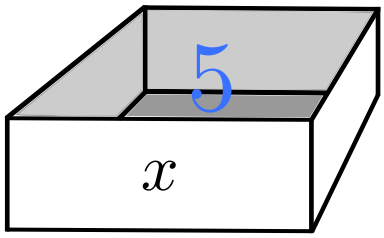

Рис. 1. Переменная

На языке программирования Python инициализация переменной делается следующим образом.

```
x = 5
```

Здесь `x` — название переменной (то есть метка на нашей коробке), число `5` — это содержимое переменной, а символ `=` означает, что мы положили число `5` в переменную с названием `x`.

Теперь перейдём к рассмотрению функции. Функцию можно представить в виде торгового автомата. Автомат принимает на вход денежные купюры, а на выходе выдаёт

тот товар, который вы выбрали. То есть автомат обрабатывает вашу информацию на входе, и выдаёт что-то на выходе.

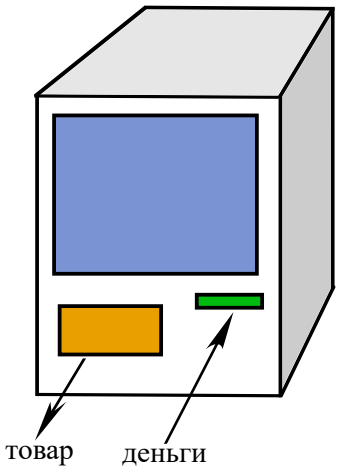

Рис. 2. Функция

Для работы нам понадобятся две функции, позволяющие взаимодействовать с пользователем. У любой функции есть своё название и список аргументов (информация на входе). Так выглядит синтаксис обращения к функции.

```
function_name(arguments)
```

Функция `print()` принимает на вход какую-то строку, а в качестве выходной информации выводит введённую строку на экран.

```
x = 5
print(5)

Out[]: 5
```

Функция `input()` принимает на вход информацию, введённую пользователем с клавиатуры. Поскольку пользователь сам вводит эту информацию, то в скобках мы ничего не указываем. А в качестве результата работы функции выдаётся то, что ввёл пользователь.

```
s = input()
print(s)

In[]: Hello world!
Out[]: Hello world!
```

Здесь пользователь вводит что-то с клавиатуры. Этот ввод сохраняется в переменную `s`. А затем содержимое переменной `s` выводится на экран.

2. АРИФМЕТИКА

2.1. Базовые арифметические операции

Компьютер умеет работать только с числами. Вся информация: текст, изображения, видео, звуки — всё это хранится в виде чисел. Поэтому важно научиться работать с базовой арифметикой.

Сумма чисел.

```
x = 5
y = 2
z = x + y
print(z)

Out[]: 7
```

Разность двух чисел.

```
x = 5
y = 2
z = x - y
print(z)

Out[]: 3
```

Произведение двух чисел.

```
x = 5
y = 2
z = x * y
print(z)

Out[]: 10
```

Остаётся ещё одна базовая операция — это операция деления. Но в Python существует целых три операции деления. Первая — обыкновенное деление.

```
x = 5
y = 2
z = x / y
print(z)

Out[]: 2.5
```

Вторая операция деления — целочисленное деление. Данная операция находит целую часть от деления.

```
x = 5
y = 2
z = x // y
print(z)

Out[]: 2
```

Третья операция деления — нахождение остатка от деления.

```
x = 5
y = 2
z = x % y
print(z)

Out[]: 1
```

Ещё одна полезная арифметическая операция — возведение в степень.

```
x = 5
y = 2
z = x ** y
print(z)

Out[]: 25
```

2.2. Приоритет операций

Также как и в математике, в языке программирования Python арифметические операции обладают различным приоритетом.

1. Возведение в степень **

2. Умножение/Деление `*, /, //, %`

3. Сложение/Вычитание `+, -`

Операции, находящиеся на одном приоритетном уровне выполняются по порядку слева направо.

```
x = 5
y = 2
z = 3
w = x ** y + y * z // x
print(w)

Out[]: 26
```

Несложно проверить корректность полученного результата. `x ** y` будет равно $5^2 = 25$. Следующий по приоритету идёт блок `y * z // x` который выполняется слева направо $2 \cdot 3 \div 5 = 1$. И чтобы посчитать значение итогового выражения остаётся сложить полученные результаты $25 + 1 = 26$.

Если есть потребность в изменении приоритета операций можно использовать круглые скобки `(...)`. Тогда вычисления будут производится по следующим правилам.

1. Если в выражении есть скобки, то сначала вычисляется значение внутри скобок.

2. Если в выражении нет скобок, то вычисления происходят согласно приоритету арифметических операций.

```
x = 5
y = 2
z = 3
w = (x ** y + y) * z // x
print(w)

Out[]: 16
```

Рассмотрим принцип вычислений на этом примере. В данном выражении есть скобки, а именно `(x ** y + y)`, поэтому сначала вычисляем значение внутри скобок согласно приоритету арифметических операций $5^2 + 2 = 25 + 2 = 27$. Других скобок нет, значит вычисления производим согласно приоритету арифметических операций $27 \cdot 3 \div 5 = 81 \div 5 = 16$.

3. ТИПЫ ДАННЫХ

В любом языке программирования информация может быть представлена разными типами данных.

- `int` — целые числа. Например, `9, -4, 0`.
- `float` — числа с плавающей запятой (вещественные числа). Например, `0.2, 2.33333333, -4.1956019, 1.0`.
- `char` — символы. Например, `'a', 'x', '6', '+'`.
- `string` — строки. Например, `'text', '6.23', 'Hello world!'`.

Разница заключается в принципе выделения памяти и принципах обработки значений.

Рассмотрим задачу: пользователь вводит два числа. А программа считает сумму этих чисел. Казалось бы, задача решается довольно легко. Сначала считаем значения, введённые пользователем с помощью функции `input()`. После чего найдём сумму и выведем на экран, используя функцию `print()`.

```
a = input()
b = input()
c = a + b
print(c)

In[]: 2
 ...: 3

Out[]: 23
```

Получается странный результат. Ибо вместо нахождения суммы двух чисел программа просто написала их рядом. В чём же заключается проблема? Суть в том, что результатом работы функции `input()` является строка (то есть тип `string`). А мы хотим получить число. Для того, чтобы преобразовать строку в числу существует функция `int()`. Функция `int()` принимает на вход строку, а на выходе выдаёт число, если она может преобразовать строку в число.

```
a_s = input()
b_s = input()
a = int(a_s)
b = int(b_s)
c = a + b
print(c)

In[]: 2
 ...: 3

Out[]: 5
```

Написанный код получился довольно громоздким для такой, кказалось бы, простой операции. Одним из основных преимуществ языка программирования Python заключается в том, что многие операции можно реализовать довольно компактным образом. В Python функции можно вкладывать друг в друга. Например, процесс обработки данных вложенными функциями для команды `int(input())` представлен на схеме ниже. Сначала выполняется внутренняя функция `input()`. Затем результат работы этой

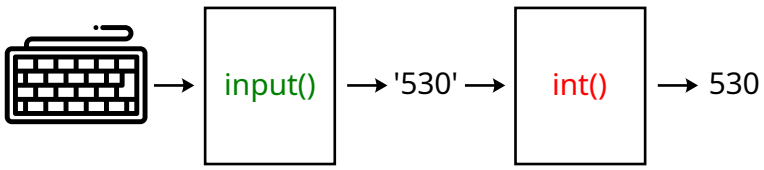

Рис. 3. Вложенные функции

функции автоматически передаётся во внешнюю функцию `int()`.

Тогда решение исходную задачу можно решить следующим образом.

```
a = int(input())
b = int(input())
c = a + b
print(c)

In[]: 2
 ...: 3

Out[]: 5
```

4. КРАСИВЫЙ ВЫВОД

4.1. Разделитель sep

Пусть у нас имеется несколько переменных. И нам нужно вывести значения всех этих переменных на экран.

```
x = 'First'
y = 'Second'
z = 'Third'
w = 'Fourth'
print(x)
print(y)
print(z)
print(w)

Out[]: First
  ...: Second
  ...: Third
  ...: Fourth
```

Недостаток такого решения заключается в том, что нам приходится вызывать функцию `print()` целых 4 раза! На самом деле Python позволяет вывести все нужные нам значения с одним вызовом функции `print()`. Идея заключается в том, что мы можем перечислить нужные нам переменные через запятую.

```
x = 'First'
y = 'Second'
z = 'Third'
w = 'Fourth'
print(x, y, z, w)

Out[]: First Second Third Fourth
```

Но здесь мы сталкиваемся с другой проблемой. Все они теперь записаны в одну строчку. А если мы хотим каждую переменную выводить с новой строки? И это можно сделать. Суть в том, что функция выводит на экран значения переменных черех пробел. В данном случае пробел является разделительным символом. Этот разделительный символ мы можем поменять. У функции `print()` есть параметр `sep` (от англ. separator — разделитель). После перечисления всех переменных через запятую нужно записать следующую конструкцию: `sep='x'`, где `x` — разделительный символ.

```
x = 'First'
y = 'Second'
z = 'Third'
w = 'Fourth'
print(x, y, z, w, sep='_')

Out[]: First_Second_Third_Fourth
```

Что интересно, в качестве разделителя можно использовать не только символ, но даже целую строку!

```
x = 'First'
y = 'Second'
z = 'Third'
w = 'Fourth'
print(x, y, z, w, sep='_nice_')

Out[]: First_nice_Second_nice_Third_nice_Fourth
```

А как всё-таки сделать перенос на новую строчку? Оказывается переход на новую строчку также является символом. Существует целый класс, так называемых специальных символов. Все специальные символы начинаются с символа \, после которого идёт код символа. Ниже приведены некоторые из этих символов.

Символ	**Назначение**
\n	Перенос вывода на новую строку
\t	Табуляция (4 пробела подряд одним символом)
\a	Звуковой сигнал

```
x = 'First'
y = 'Second'
z = 'Third'
w = 'Fourth'
print(x, y, z, w, sep='\n')

Out[]: First
  ...: Second
  ...: Third
  ...: Fourth
```

4.2. f-строка

Язык программирования Python предоставляет ещё один мощный инструмент, позволяющий делать красивый вывод. Пусть даны две переменные, содержащие числа. Необходимо вывести на экран их сумму, но в формате `3 + 2 = 5`.

```
x = 3
y = 2
z = x + y
print(x, '+', y, '=', z)

Out[]: 3 + 2 = 5
```

В целом, это довольно неплохое решение. Но что если, нам нужно будет убрать некоторые пробелы. Например, `3+2 = 5`.

```
x = 3
y = 2
z = x + y
print(x, '+', y, ' = ', z, sep='')

Out[]: 3+2 = 5
```

Здесь в качестве разделителя был поставлен пустой символ `''`. Но тогда слипнется всё. Но нам нужно, чтобы знак `=` отделялся пробелами. Поэтому они были указаны в явном виде. Изначальная задача не сильно изменилась, а вот решение стало уже более неприятным. Чтобы сделать этот процесс более приятным, можно использовать так называемые `f`-строки (от англ. formatted — форматированная). Чтобы описать `f`-строку, нужно поставить символ `f` перед кавычками.

```
print(f'Hello world!')

Out[]: Hello world!
```

Пока ничего не изменилось. Но внутрь `f`-строк можно вставлять значения переменных. Для этого название переменной нужно обернуть в фигурные скобки.

```
x = 5
print(f'The variable stores a number {x}')

Out[]: The variable stores a number 5
```

Тогда исходную задачу можно решить так.

```
x = 3
y = 2
z = x + y
print(f'{x} + {y} = {z}')

Out[]: 3 + 2 = 5
```

Основное преимущество заключается в том, что мы можем сразу писать столько символов в разных местах, сколько нам нужно, не думая о разделителе.