Часть 1. Переменные и типы данных

1. Добро пожаловать в Летающий Цирк.

Python – это мощный и гибкий язык программирования, который можно использовать для разработки web-приложений, для написания декстопных программ с графическим интерфейсом (GUI), на нём можно разрабатывать игры и многое другое.

Python это:

- язык программирования высокого уровня, то есть чтение кода на Python и написание кода на нём очень простое – он весьма схож с обычным английским языком;
- интерпретируемый – то есть, вам не потребуется специальный компилятор скриптов, чтобы писать на Python и запускать его скрипты;
- он объектно-ориентированный, то есть – позволяет пользователям управлять структурами данных, называемыми “объекты“, для построения и выполнения программ, мы обсудим объекты позже, в нашем курсе.;
- весёлый в изучении и использовании. Python получил своё название в честь Летающего Цирка Монти Пайтона (Monty Python’s Flying Circus), и примеры в коде и учебники часто ссылаются на это шоу и включают в себя шутки, что бы сделать изучения языка более интересным.

Этот курс не предполагает каких-либо знаний у вас о Python в частности, или о программировании в целом.

2. Переменные

Одним из наиболее важных понятий в программировании является понятие “переменная“. Переменными называют слово/идентификатор, которое содержит единственное значение. Например, предположим что вам требуется число 5 в вашей программе, но вы не хотите использовать его немедленно. Вы можете назначить переменную, назовём её “spam“ которая имеет значение “5” и хранит его для использования в дальнейшем:

```
>>> spam = 5
```

Объявить переменную в Python очень просто – просто указываете имя/идентификатор, как spamи используете знак = для присвоения ей значения – всё готово!

Давайте зададим переменной `my_variable` значение 10, и посмотрим его вывод:

```
>>> my_variable = 10
>>> my_variable
10
```

3. Типы данных

Отлично, теперь мы можем вызывать значение 10 просто вызывая переменную `my_variable` где угодно, когда нам потребуется.

В этом случае, тип данных переменной `my_variable` является integer (положительное или отрицательное целое число). В Python на сегодняшний день существует три типа данных: integers, floats (дробные числительные, например 1.970) и booleans (логический тип данных, который может принимать значение либо `True` (истина), либо `False` (ложно)).

Компьютерные программы, по большей части, созданы для обработки данных. Поэтому очень важно для вас понимать разницу между разными типами данных, которые мы используем в наших программах.

Никогда не используйте кавычки (‘ или “) при определении логических `True` или `False`, и всегда указывайте их с Большой буквы. Python – регистро-зависимый язык, и значения для `true, truE`или `True` будут совершенно разными. Что касается кавычек, мы начнём использовать их, когда дойдём до изучения `последовательностей (strings)`, которые будут обсуждать в следующих частях.

Давайте установим такие значения нашим переменным:

```
    1.my_int to the value 7
    2.my_float to the value 1.23
    3.my_bool to the value True
Python detected the following datatypes:
my_int is of <type ‘int’>
my_float is of <type ‘float’>
my_bool is of <type ‘bool’>
```

4. Переназначение данных

Теперь вы знаете как объявлять переменные и присваивать им разные значения. Так же, вы узнали о трёх типах данных – integers, floats, и booleans.

Вы можете переназначить значение переменной в любое время. Если сначала вы установили для переменной `my_int` значение 7, но потом решили изменить его на 3 – просто скажите Python-у: `my_int = 3` – и он изменит значение этой переменной:

```
>>> my_int = 7
>>> my_int
7
>>> my_int = 3
```

```
>>> my_int
3
```

Часть 2. Пробелы и выражения

1. Что такое операторы

Вы можете думать о выражениях (statements) Python-а как о простых фразах в английском: это наименьшие объекты в языке, которые имеют смысл. Как, например – “Я“, “люблю” и “Спам” не являются простыми выражениями сами по себе и не несут смысловой нагрузки, но “Я люблю спам” – это уже фраза, имеющая определённое значение. Так же переменные и типы данных не являются выражениями сами по себе, но они являются “строительными блоками”, из которых формируются выражения.

Продолжая аналогию с выражения ясно, что нам нужны определённые правила пунктуации, что бы определить где заканчивается одно выражения и начинается другое. Если вы знакомы с JavaScript, вы знаете что в нём операторы заканчиваются символом точки с запятой – ” ; “.

В Python операторы разделяются пробелами. Так же, как вы не можете в JavaScript вынести часть кода за пределы ограничивающих его точки с запятой, так же не сможете обойти пробел в Python-е.

Возможно, к этому придётся привыкнуть, особенно – если вы начали изучение Python после изучения языка, в котором пробелы не играют роли.

Возьмём, например, скрипт с таким кодом:

```
def spam():
eggs = 12
return eggs
print spam()
```

При его выполнении – мы получим сообщени от интерпретатора Python-а об ошибке:

```
File “./codeacademy_example_1.py”, line 5
eggs = 12
^
IndentationError: expected an indented block
```

Вот само описание: “expected an indented block” – ожидаются отступы блока.

2. Пробелы – значит Правильные отступы

Запомните ошибку, которую вы получили в предыдущем примере: “ожидаются отступы блока“. Вы будете получать такое же сообщение везде, где в коде Python вы будете неправильно использовать пробелы. Если вы ранее изучали JavaScript – думайте о неправильном использовании пробелов в Python как о неправильном использовании

символов ;или {} в JS. Если вы используете неправильные знаки препинания, смысл ваших слов может совершенно измениться:

> *Крестьянин сказал – "Ведьма превратила меня в лягушку!".*

Или:

> *"Крестьянин, – сказала ведьма, – превратил меня в лягушку!".*

Понимаете, что мы имеем ввиду?

Давайте правильно расставим отступы в коде с помощью клавиши табуляции Tab на вашей клавиатуре – один раз на второй строке перед `eggs`, и ещё раз на строке 3 перед `return`:

```
def spam():
eggs = 12
return eggs
print spam()
```

3. Про интерпретатор

Во введении в эту главу мы писали, что Python – интерпретируемый язык, то есть – он обрабатывается `интерперетатором`.

Так, вы можете думать об интерпретаторе, как о программме, которая получает вводимый вами код, проверяет его на наличие ошибок, выполняет операции в этом коде, строка за строкой. Интерпретатор работает в фоновом режиме, невидимо для вас и отбражает только рузультат ваших инструкций машине.

Давайте дадим задачу интепретатору назначить значение `True` для переменной `spam`, и `False` – для `eggs`:

```
>>> spam = True
>>> eggs = False
>>> spam
True
>>> eggs
False
>>> print spam, eggs
True False
```

Часть 3. Комментарии

1. Однострочные комментарии

Вы можете добавлять заметки в ваш код, устанавливая символ # (hash) в начале строки. Эти строки в коде называются комментариями, и они игнорируются интерпретатором – они совершенно не влияют на исполнение кода. Это просто текстовые замечания, написанные программистами что бы дать некоторые инструкции или пояснения к конкретной части программы.

Так как это улучает читаемость кода, и поможет в отладке, так как вы сможете быстро сказать – какая часть программы что именно должна выполнять, мы призываем вас добавлять комментарии везде, где результат выполнения части кода не является очевидным.

2. Многострочные комментарии

Иногда требуется написать большой комментарий, который не поместится в одну строку. Вы можете, конечно, написать комментарии в несколько строк, начиная каждую их них символом # – но это, согласитесь, лишняя головная боль.

Для написания мультистрочных комментариев – используйте тройные кавычки, как в примере ниже:

```
“””Я программист, и я счастлив.
Я ем Мивину и запиваю её чаем!”””
```

Часть 4. Математические операции

1. Арифметические операторы

Выражения в Python не ограничены только простыми выражениями типа `spam = 3`. Они могут так же включать в себя математические выражения, использующие `арифметические операторы (arithmetic operators)`.

Есть шесть таких арифметических операторов, которые мы и будем рассматривать:

1. `Сложение (Addition) (+)`
2. `Вычитание (Subtraction) (-)`
3. `Умножение (Multiplication) (*)`
4. `Деление (Division) (/)`
5. `Возведение в квадрат (Exponentiation) (**)`
6. `Деление с остатком (Modulo) (%)`

Давайте начнём со сложения. Установим значение переменной `big_var` равным сумме 1 + 2:

```
>>> big_var = 3 + 2
>>> big_var
5
```

2. Вычитание (Subtraction)

Теперь – попробуем вычитание.

Мы уже посчитали, что значение переменной `big_var` равняется 5, но теперь хотим уменьшить его. Укажем интерпретатору переназначить значение переменной, выполнив вычитание 5 – 2:

```
>>> big_var = 5 - 2
>>> big_var
3
```

3. Умножение (Multiplication)

Теперь – давайте попробуем выполнить умножение.

Рыцарь, который сказал “Ni!” сказал это дважды. Давайте сделаем это 20 раз, умножив 2 * 10:

```
>>> ni = 2 * 10
>>> ni
20
```

4. Деление (Division)

Немного подумав, мы решили что 20 “Ni!” будет слишком много. Давайте используем деление, что бы уменьшить значение до 5:

```
>>> ni = 20 / 4
>>> ni
5
```

5. Возведение в степень (Exponentiation)

Все арифметические операции, которые вы сделали до сих пор либо были интуитивно понятны или напоминали действия, которые вам доводилось выполнять в других языках программирования (например, JavaScript). Возведение в степень, однако, является более сложной операцией, так что мы остановимся на ней подробнее.

Арифметический оператор ** возводит первое число – base, в значение второго числа – exponent. Так, если вы наберёте 2 ** 3 – вы получите 8 (2 ** 3 это тоже самое, что 2 * 2 * 2 – обе операции дадут значение 8). 5 ** 2 = 25, 2 ** 4 = 16 и так далее.

Итак, наш дровосек очень голоден, и хочет съесть сразу 100 яиц. Давайте установим значение переменной eggs равным 100, используя Exponentiation:

```
>>> eggs = 10**2
>>> eggs
100
```

6. Деление с остатком (Modulo)

Теперь у нас множество яиц. Надеюсь – вы оставили место в комнате для `spam`? (что тут имеется ввиду – затрудняюсь сказать… вероятно – мясные консервы производителя Spam – прим. перевод.)

Наша последняя арифметическая операция – деление с остатком, modulo (так же называемая modulus). Modulo возвращает остаток, полученный в результате деления (division) двух целых (integer) чисел.

Так, если вы попросите интерпретатор посчитать значение 5 % 2 – вы получите результат 1 (т.к. 2 входит в 5 дважды и даёт остаток 1). 10 % 5 = 0, 113 % 100 = 13 и так далее. Давайте у становим значение переменной `spam` равным 1, используя modulo:

```
>>> spam = 45 % 22
>>> spam
1
```

Часть 5. Заключение.

1. Соберём всё в одном

Итак, вы изучили такие понятия в Python:

- Переменные, которые заключают в себе значения;
- Типы данных, такие как integers, floats, и booleans;
- Пробелы, и почему они так важны;
- Операнды, и чем они схожи с фразами в английском языке;
- Комментарии, и почему они так нужны в вашем коде;
- Арифметические действия и операторы, такие как +, -, *, /, **, и %)

Давайте соберём все наши знания в одно целое, и заставим это всё поработать.

- напишем однострочный комментарий;
- объявим переменную monty и установим её значение равным True;
- объявим вторую переменную, и устанвоим её значеине раыным 1.234;
- объявим третью переменную, и установим её значение равнывм python в степени 2.

```
>>> monty = True
>>> python = 1.234
>>> monty_python  = python ** 2
>>> monty_python
1.522756
```

Закрепление знаний

1. Ваша любимая еда.

Этот раздел написан специально для закрепления материала первого раздела – Python Syntax.

Итак, вы только что заказали прекрасную порцию вкусного мяса (`spam`) и яиц (`eggs`).

Вот как это выглядит в цифрах:

Стоимость еды (`meal`): $44.50

Надбавка ресторана (`tax`): 6.75%

Чаевые (`tip`): 15%

Вы хотите дать чаевые, учитывая полную стоимость еды, включая надбавку ресторана.

Для начала, давайте зададим значение переменной `meal` равным 44.50:

```
>>> meal = 44.50
>>> meal
44.5
```

2. Ресторанная надбавка

Теперь – давайте создадим значение для переменной надбавка.

Надбавка стоимости вашей еды составит 6.75%. Так как мы будем иметь дело с дробными числами, а не с процентами, давайте переведём это значение в float, т.е. дробное десятичное число. Для этого – поделим 6.75 на 100, в результате мы получим обычное десятичное число.

Создадим переменную `tax` и установим её значение равным десятичному значению 6.75%:

```
>>> tax = 6.75 / 100
>>> tax
0.0675
```

3. Чаевые

Вам понравилось обслуживание, и и вы хотите оставить 15% сверх общей цены мяса и надбавки в качестве чаевых.

Перед тем, как с читать общую стоимость, давайте установим значение для переменной `tip` в размере 15%. Опять-таки, эта цифра в процентах – так что нам надо поделить 15.0 на 100, чтобы получить десятичное значение:

```
>>> meal = 44.50
>>> tax = 0.0675
>>> tip = 15.0 / 100
>>> tip
0.15
```

4. Переназначем значение в одну строку

Итак, мы получили три переменных, которые нам нужны для наших вычислений и у нас есть несколько арифметических операторов, которые нам в этом помогут.

Мы видели в предыдущих уроках, что мы можем переназначать значение переменной когда нам потребуется. Например, мы можем установить значение переменной `spam` = 7 в одном месте программы, и позже изменить его на другое, указав `spam` = 3.

Давайте переназначим значение переменной `meal` на значение `meal + meal * tax`. Вы можете менять значение переменной, применяя в новом значении её старое значение:

```
>>> meal = 44.50
>>> tax = 0.0675
>>> tip = 0.15
>>> meal = 44.50 + 44.50 * 0.0675
>>> print meal
47.50375
```

5. Второй куплет похож на первый

Почти всё готово. Теперь мы выполним почти то же самое, только на этот раз мы введём новую переменную, вместо того, что бы сбрасывать и назначать новое значение старой.

Теперь, когда значение переменной `meal` равно стоимости `food + tax`, давайте зададим новую переменную `total`, которая будет содержать значение равное `meal + meal * tip`.

Это действие очень похоже на выполненное только что.

Наш код красиво отформатирует значение переменной `total` и выведет результат на консоль с ограничением в два знака после запятой (мы поговорим о форматировании строк, о консоли, и о форме `print` в следующих занятиях).

Установим значение `total` в качестве суммы `meal + meal * tip`:

```
>>> meal = 44.50
>>> tax = 0.0675
>>> tip = 0.15
>>> meal = meal + meal * tax
>>> total = meal + meal * tip
>>> print(“%.2f” % total)
54.63
```