Введение в Python. Целые и вещественные числа. Логические переменные.

Функция print()

С помощью Python можно решать огромное количество задач. Мы начнем с очень простых и постепенно будем их усложнять, закончив наш курс небольшим проектом. Если вы уже сталкивались с программированием, то вы помните, что обычно самой первой программой становится вывод "Hello, world". Попробуем сделать это в Python.

In [1]:
```
print('Hello, world!')
print(1)
```

```
Hello, world!
1
```

Обратите внимание, что "Hello, world!" мы написали в кавычках, а единицу - нет. Это связанно с тем, что в программировании мы имеем дело с разными типами данных. И Python будет воспринимать текст как текст (строковую переменную), только в том случае, если мы его возьмем в кавычки (неважно, одинарные или двойные). А при выводе эти кавычки отображаться уже не будут (они служат знаком для Python, что внутри них - текст).

print() - это функция, которая выводит то, что мы ей передадим. В других IDE это был бы вывод в терминал, а в Jupyter вывод напечатается под запускаемой ячейкой. Распознать функцию в питоне можно по скобкам после слова, внутри которых мы передаем аргумент, к которому эту функцию нужно применить (текст "Hello, world" или 1 в нашем случае).

In [2]:
```
print(Hello, world)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-65c2bc1cadaf> in <module>()
----> 1 print(Hello, world)

NameError: name 'Hello' is not defined
```

Написали без кавычек - получили ошибку. Кстати, обратите внимание, что очень часто в тексте ошибки есть указание на то, что именно произошло, и можно попробовать догадаться, что же нужно исправить. Текст без кавычек Python воспринимает как название переменной, которую еще не задали. Кстати, если забыть закрыть или открыть кавычку (или поставить разные), то тоже поймаем ошибку.

Иногда мы хотим комментировать наш код, чтобы я-будущий или наши коллеги поменьше задавались вопросами, а что вообще имелось ввиду. Комментарии можно писать прямо в коде, они не повлияют на работу программы, если их правильно оформить.

In [5]:
```
# Обратите внимание: так выглядит комментарий - часть скрипта, которая не будет испо
# при запуске программы.
# Каждую строку комментария мы начинаем со знака хэштега.

'''
Это тоже комментарий - обычно выделение тремя апострофами мы используем для тех случ
```

```
когда хотим написать длинный, развернутый текст.
'''

print('Hello, world')
```

```
Hello, world
```

Обратите внимание, что в отличие от других IDE (например, PyCharm) в Jupyter Notebook не всегда обязательно использовать print(), чтобы что-то вывести. Но не относитесь к этому как к стандарту для любых IDE.

In [1]:
```
"Hello, world"
```

Out[1]:
```
'Hello, world'
```

Выше рядом с выводом появилась подпись Out[]. В данном случае Jupyter показывает нам последнее значение, лежащее в буфере ячейки. Например, в PyCharm такой вывод всегда будет скрыт, пока мы его не "напечатаем" с помощью print(). Но эта особенность Jupyter помогает нам быстро проверить, что, например, находится внутри переменной, когда мы отлаживаем наш код.

Следующая вещь, которую нужно знать про язык программирования - как в нем задаются переменные. Переменные - это контейнеры, которые хранят в себе информацию (текстовую, числовую, какие-то более сложные виды данных). В Python знаком присвоения является знак =.

In [1]:
```
x = 'Hello, world!'
print(x)    # Обратите внимание, что результат вызова этой функции такой же, как выш
            # только текст теперь хранится внутри переменной.
```

```
Hello, world!
```

Python - язык чувствительный к регистру. Поэтому, когда создаете/вызываете переменные или функции, обязательно используйте правильный регистр. Так, следующая строка выдаст ошибку.

In [2]:
```
print(X) # мы создали переменную x, а X не существует
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-f9979d7199c5> in <module>()
----> 1 print(X) # мы создали переменную x, а X не существует

NameError: name 'X' is not defined
```

Еще раз обратим внимание на ошибку. *NameError: name 'X' is not defined* означает, что переменная с таким названием не была создана в этой программе. Кстати, обратите внимание, что переменные в Jupyter хранятся на протяжении всей сессии (пока вы работаете с блокнотом и не закрыли его), и могут быть созданы в одной ячейке, а вызваны в другой. Давайте опять попробуем обратиться к x.

In [11]:
```
print(x) # работает!
```

```
Hello, world!
```

Типы данных: целочисленные переменные (integer)

Знакомство с типа данных мы начнем с целых чисел. Если вы вдруг знакомы с другими языками программирования, то стоит отметить, что типизация в Python - динамическая. Это значит, что вам не нужно говорить какой тип данных вы хотите положить в переменную - Python сам все определит. Проверить тип данных можно с помощью функции type(), передав ей в качестве аргумента сами данные или переменную.

ЦЕЛЫЕ ЧИСЛА (INT, INTEGER): 1, 2, 592, 1030523235 - любое целое число без дробной части.

In [3]:
```
y = 2
print(type(2))
print(type(y))
```

```
<class 'int'>
<class 'int'>
```

Обратите внимание - выше мы вызвали функцию внутри функции. type(2) возвращает скрытое значение типа переменной (int для integer). Чтобы вывести это скрытое значение - мы должны его "напечатать".

Самое элементарное, что можно делать в Python - использовать его как калькулятор. Давайте посмотрим, как он вычитает, складывает и умножает.

In [14]:
```
print(2 + 2)
print(18 - 9)
print(4 * 3)
```

```
4
9
12
```

С делением нужно быть немного осторожней. Существует два типа деления - привычное нам, которое даст в ответе дробь при делении 5 на 2, и целочисленное деление, в результате которого мы получим только целую часть частного.

In [4]:
```
print(5 / 2)  # в результате такого деления получается другой тип данных (float), по
print(5 // 2)
```

```
2.5
2
```

А если нам надо как раз найти остаток от деления, то мы можем воспользоваться оператором модулю %

In [17]:
```
print(5 % 2)
```

```
1
```

Еще одна математическая операция, которую мы можем выполнять без загрузки специальных математических библиотек - это возведение в степень.

In [5]:
```
print(5**2)
```

```
25
```

Все то же самое работает и с переменными, содержащими числа.

In [19]:
```
a = 2
b = 3
print(a ** b)
# изменится ли результат, если мы перезапишем переменную a?
a = 5
print(a ** b)
```

```
8
125
```

Часто возникает ситуация, что мы считали какие-то данные в формате текста, и у нас не работают арифметические операции. Тогда мы можем с помощью функции int() преобразовать строковую переменную (о них ниже) в число, если эта строка может быть переведена в число в десятичной системе.

In [6]:
```
print(2 + '5') # ошибка, не сможем сложить целое число и строку
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-f28303544e9d> in <module>()
----> 1 print(2 + '5') # ошибка, не сможем сложить целое число и строку

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

In [7]:
```
print(2 + int('5')) # преобразовали строку в число и все заработало
```

```
7
```

In [8]:
```
int('текст') # здесь тоже поймаем ошибку, потому что строка не представляет собой чи
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-8-5879db2a084c> in <module>()
----> 1 int('текст') # здесь тоже поймаем ошибку, потому что строка не представляет
 собой число

ValueError: invalid literal for int() with base 10: 'текст'
```

(∩ ` -´)⊃–☆゚.*･｡ﾟ Задача

Сумма цифр трехзначного числа

Дано трехзначное число 179. Найдите сумму его цифр.

Формат вывода

Выведите ответ на задачу.

Ответ

Вывод программы:

```
17
```

In [9]:
```
# (∩ `-´)⊃━☆ﾟ.*･｡ﾟ

x = 179
x_1 = x // 100
x_2 = x // 10 % 10
x_3 = x % 10
print(x_1, x_2, x_3) # тестовый вывод, проверяем, что правильно "достали" цифры из ч
print(x_1 + x_2 + x_3) # ответ на задачу
```

```
1 7 9
17
```

(∩ `-´)⊃━☆ﾟ.*･｡ﾟ Задача

Электронные часы

Дано число N. С начала суток прошло N минут. Определите, сколько часов и минут будут показывать электронные часы в этот момент.

Формат ввода

Вводится число N — целое, положительное, не превышает 10^7.

Формат вывода

Программа должна вывести два числа: количество часов (от 0 до 23) и количество минут (от 0 до 59).

Учтите, что число N может быть больше, чем количество минут в сутках.

Примеры

Тест 1
Входные данные:
150

Вывод программы:
2 30

In [2]:
```
# (∩ `-´)⊃━☆ﾟ.*･｡ﾟ

minutes = 150

print(minutes // 60 % 24, minutes % 60)
```

```
2 30
```

Типы данных: логические или булевы переменные (boolean)

Следующий тип данных - это логические переменные. Логические переменные могут принимать всего два значения - **истина (True)** или **ложь (False)**. В Python тип обозначается **bool**.

In [12]:
```
print(type(True), type(False))
```

```
<class 'bool'> <class 'bool'>
```

Логические переменные чаще всего используется в условных операторах if-else и в цикле с остановкой по условию while. В части по анализу данных еще увидим одно частое применение - создание булевых масок для фильтрации данных (например, вывести только те данные, где возраст больше 20).

Обратите внимание, что True и False обязательно пишутся с большой буквы и без кавычек, иначе можно получить ошибку.

In [22]:
```
print(type('True')) # тип str - строковая переменная
print(true) # ошибка, Python думает, что это название переменной
```

```
<class 'str'>
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-22-7baca652a4cb> in <module>()
      1 print(type('True')) # тип str - строковая переменная
----> 2 print(true) # ошибка, Python думает, что это название переменной

NameError: name 'true' is not defined
```

Как и в случае чисел и строк, с логическими переменными работает преобразование типов. Превратить что-либо в логическую переменную можно с помощью функции bool().

Преобразование чисел работает следующим образом - 0 превращается в False, все остальное в True.

In [33]:
```
print(bool(0))
print(bool(23))
print(bool(-10))
```

```
False
True
True
```

Пустая строка преобразуется в False, все остальные строки в True.

In [34]:
```
print(bool(''))
print(bool('Hello'))
print(bool(' ')) # даже строка из одного пробела - это True
print(bool('False')) # и даже строка 'False' - это True
```

```
False
True
True
True
```

И при необходимости булеву переменную можно конвертировать в int. Тут все без сюрпризов - ноль и единица.

In [23]:
```
print(int(True))
print(int(False))
```

```
1
0
```

Логические выражения

Давайте посмотрим, где используется новый тип данных.

Мы поработаем с логическими выражениями, результат которых - булевы переменные.

Логические выражения выполняют проверку на истинность, то есть выражение равно True, если оно истинно, и False, если ложно.

В логических выражениях используются операторы сравнения:

- == (равно)
- != (не равно)
- > (больше)
- < (меньше)
- >= (больше или равно)
- <= (меньше или равно)

In [24]:
```
print(1 == 1)
print(1 != '1')
c = 1 > 3
print(c)
print(type(c))
x = 5
print(1 < x <= 5) # можно писать связки цепочкой
```

```
True
True
False
<class 'bool'>
True
```

Логические выражения можно комбинировать с помощью следующих логических операторов:

- логическое И (and) - выражение истинно, только когда обе части истинны, иначе оно ложно
- логическое ИЛИ (or) - выражение ложно, только когда обе части ложны, иначе оно истинно
- логическое отрицание (not) - превращает True в False и наоборот

In [37]:
```
print((1 == 1) and ('1' == 1))
print((1 == 1) or ('1' == 1))
print(not(1 == 1))
print(((1 == 1) or ('1' == 1)) and (2 == 2))
```

```
False
True
False
True
```

(∩ ` -´)⊃–☆゚.*･｡ﾟ Задача

Вася в Италии

Вася уехал учиться по обмену на один семестр в Италию. Единственный магазин в городе открыт с 6 до 8 утра и с 16 до 17 вечера (включительно). Вася не мог попасть в магазин уже несколько дней и страдает от голода. Он может прийти в магазин в X часов. Если магазин открыт в X часов, то выведите True, а если закрыт - выведите False.

В единственной строке входных данных вводится целое число X, число находится в пределах от 0 до 23

Формат ввода

Целое число X, число находится в пределах от 0 до 23

Формат вывода

True или False

Примеры

Тест 1
Входные данные:
16

Вывод программы:
True

In [39]:
```
## (∩ `-´)⊃━☆ﾟ.*･｡ﾟ

time = 16
can_visit = 6 <= time <= 8
can_visit2 = 16 <= time <= 17
print(can_visit or can_visit2)
```

```
True
```

Типы данных: вещественные числа (float)

По сути, вещественные числа это десятичные дроби, записанные через точку.
Вещественные числа в питоне обозначаются словом float (от "плавающей" точки в них).
Также могут быть представлены в виде инженерной записи: 1/10000 = 1e-05

ВЕЩЕСТВЕННЫЕ ЧИСЛА (FLOAT): 3.42, 2.323212, 3.0, 1e-05 - число с дробной частью (в том числе целое с дробной частью равной 0)

In [10]:
```
4.5 + 5
```

Out[10]:
```
9.5
```

Если у нас было действие с участие целого и вещественного числа, то ответ всегда будет в виде вещественного числа (см. выше).

Также давайте вспомним про "обычное" деление, в результате которого получается вещественное число.

In [11]:
```
print(11 / 2)
print(type(11 / 2))
print(11 // 2)
print(type(11 // 2))
```

```
5.5
<class 'float'>
5
<class 'int'>
```

С вещественными числами нужно быть осторожными со сравнениями. В связи с особенностями устройства памяти компьютера дробные числа хранятся там весьма хитро и не всегда условные 0.2 то, чем кажутся. Это связано с проблемой точности представления чисел.

Подробнее можно прочитать здесь.

In [7]:
```
0.2 + 0.1 == 0.3
```

Out[7]:
```
False
```

Наверняка, от такого равенства мы ожидали результат True, но нет. Поэтому будьте аккуратны и старайтесь не "завязывать" работу вашей программы на условия, связанные с вычислением вещественных чисел. Давайте посмотрим, как на самом деле выглядят эти числа в памяти компьютера.

In [62]:
```
print(0.2 + 0.1)
print(0.3)
```

```
0.30000000000000004
0.3
```

Числа с плавающей точкой представлены в компьютерном железе как дроби с основанием 2 (двоичная система счисления). Например, десятичная дробь

0.125
имеет значение 1/10 + 2/100 + 5/1000, и таким же образом двоичная дробь

0.001
имеет значение 0/2 + 0/4 + 1/8. Эти две дроби имеют одинаковые значения, отличаются только тем, что первая записана в дробной нотации по основанию 10, а вторая по основанию 2.

К сожалению, большинство десятичных дробей не могут быть точно представлены в двоичной записи. Следствием этого является то, что в основном десятичные дробные числа вы вводите только приближенными к двоичным, которые и сохраняются в компьютере.

Если вам совсем не обойтись без такого сравнения, то можно сделать так: сравнивать не результат сложения и числа, а разность эти двух чисел с каким-то очень маленьким числом (с таким, размер которого будет точно не критичен для нашего вычисления). Например, порог это числа будет разным для каких-то физических вычислений, где важна высокая точность, и сравнения доходов граждан.

In [46]:
```
```

```
0.2 + 0.1 - 0.3 < 0.000001
```

Out[46]:
```
True
```

Следующая проблема, с которой можно столкнуться - вместо результата вычисления получить ошибку 'Result too large'. Связано это с ограничением выделяемой памяти на хранение вещественного числа.

In [21]:
```
1.5 ** 100000
```

```
---------------------------------------------------------------------------
OverflowError                             Traceback (most recent call last)
<ipython-input-21-449e5e88bdf9> in <module>()
----> 1 1.5 ** 100000

OverflowError: (34, 'Result too large')
```

In [23]:
```
1.5 ** 1000
```

Out[23]:
```
1.2338405969061735e+176
```

А если все получилось, то ответ еще может выглядеть вот так. Такая запись числа называется научной и экономит место - она хранит целую часть числа (мантисса) и степень десятки на которую это число нужно умножить (экспонента). Здесь результатом возведения 1.5 в степень 1000 будет число 1.2338405969061735, умноженное на 10 в степень 176. Понятно, что это число очень большое. Если бы вместо знакак + стоял -, то и степень десятки была бы отрицательная (10 в -176 степени), и такое число было бы очень, очень маленьким.

Как и в случае с целыми числами, вы можете перевести строку в вещественное число, если это возможно. Сделать это можно фукнцией float()

In [63]:
```
print(2.5 + float('2.4'))
```

```
4.9
```

Округление вещественных чисел

У нас часто возникает необходимость превратить вещественное число в целое ("округлить"). В питоне есть несколько способов это сделать, но, к сожалению, ни один из них не работает как наше привычное округление и про это всегда нужно помнить.

Большинство этих функций не реализованы в базовом наборе команд питона и для того, чтобы ими пользоваться, нам придется загрузить дополнительную библиотеку math, которая содержит всякие специальные функции для математических вычислений.

In [51]:
```
import math # команда import загружает модуль под названием math
```

Модуль math устанавливается в рамках дистрибутива Anaconda, который мы использовали, чтобы установить Jupyter Notebook, поэтому его не нужно отдельно скачивать, а можно просто импортировать (загрузить в оперативную память текущей сессии). Иногда нужную

библиотеку придется сначала установить на компьютер с помощью команды !pip install -название модуля- и только потом импортировать.

Самый простой способ округлить число - применить к нему функцию int.

In [47]:
```
int(2.6)
```

Out[47]:
```
2
```

Обратите внимание, что такой метод просто обрубает дробную часть (значения выше 0.5 не округляются в сторону большего числа).

In [49]:
```
print(int(2.6))
print(int(-2.6))
```

```
2
-2
```

Округление "в пол" из модуля math округляет до ближайшего меньшего целого числа.

In [52]:
```
print(math.floor(2.6)) # чтобы использовать функцию из дополнительного модуля -
                       # нужно сначала написать название этого модуля и через точку
print(math.floor(-2.6))
```

```
2
-3
```

Округление "в потолок" работает ровно наоборот - округляет до ближайшего большего числа, независимо от значения дробной части.

In [53]:
```
print(math.ceil(2.6))
print(math.ceil(-2.6))
```

```
3
-2
```

В самом питоне есть еще функция round(). Вот она работает почти привычно нам, если бы не одно "но"...

In [54]:
```
print(round(2.2))
print(round(2.7))
print(round(2.5)) # внимание на эту строку
print(round(3.5))
```

```
2
3
2
4
```

Неожиданно? Тут дело в том, что в питоне реализованы не совсем привычные нам правила округления чисел с вещественной частью 0.5 - такое число округляется до ближайшего четного числа: 2 для 2.5 и 4 для 3.5.

Замечание по импорту функций

Иногда нам не нужна вся библиотека, а только одна функция из-за нее. Скажите, странно же хранить в опреативной памяти всю "Войну и мир", если нас интересует только пятое предложение на восьмой странице. Для этого можно воспользоваться импортом функции из библиотеки и тогда не нужно будет писать ее название через точку. Подводный камень здесь только тот, что если среди базовых команд питона есть функция с таким же именем, то она перезапишется и придется перезапускать свой блокнот, чтобы вернуть все как было.

In [32]:
```
from math import ceil
ceil(2.6) # теперь работает без math.
```

Out[32]:
```
3
```

(∩ ` -´)⊃━☆ﾟ.*･｡ﾟ Задача

Дробная часть

Дано вещественное число. Выведите его дробную часть.

Формат ввода

Вещественное число

Формат вывода

Вещественное число (ответ на задачу)

Примеры

Тест 1
Входные данные:
4.0

Вывод программы:
0.0

In [58]:
```
# (∩ `-´)⊃━☆ﾟ.*･｡ﾟ
x = 4.0
print(x - int(x))

x = 5.2
print(x - int(x))
```

```
0.0
0.20000000000000018
```