

Лекция 1

Язык программирования Python введение.

План занятия

- Что есть python и зачем его изучать
- Интерактивная строка, и выбор
- Базовые типы данных
 - неизменяемые
 - изменяемые
- Арифметические операции над числами
- Динамическая типизация
- Переменные
- Работа с вводом/выводом
- Практика

План занятия

- Что есть python и зачем его изучать
- Интерактивная строка
- Базовые типы данных
 - неизменяемые
 - изменяемые
- Арифметические операции над числами
- Динамическая типизация
- Переменные
- Работа с вводом/выводом
- Практика

Литература

Начальный уровень

- Mark Pilgrim. Dive into Python - <http://www.diveintopython.net/>
- Марк Лутц. Изучаем Python, 4-е издание // Символ-Плюс 2011.
- ...

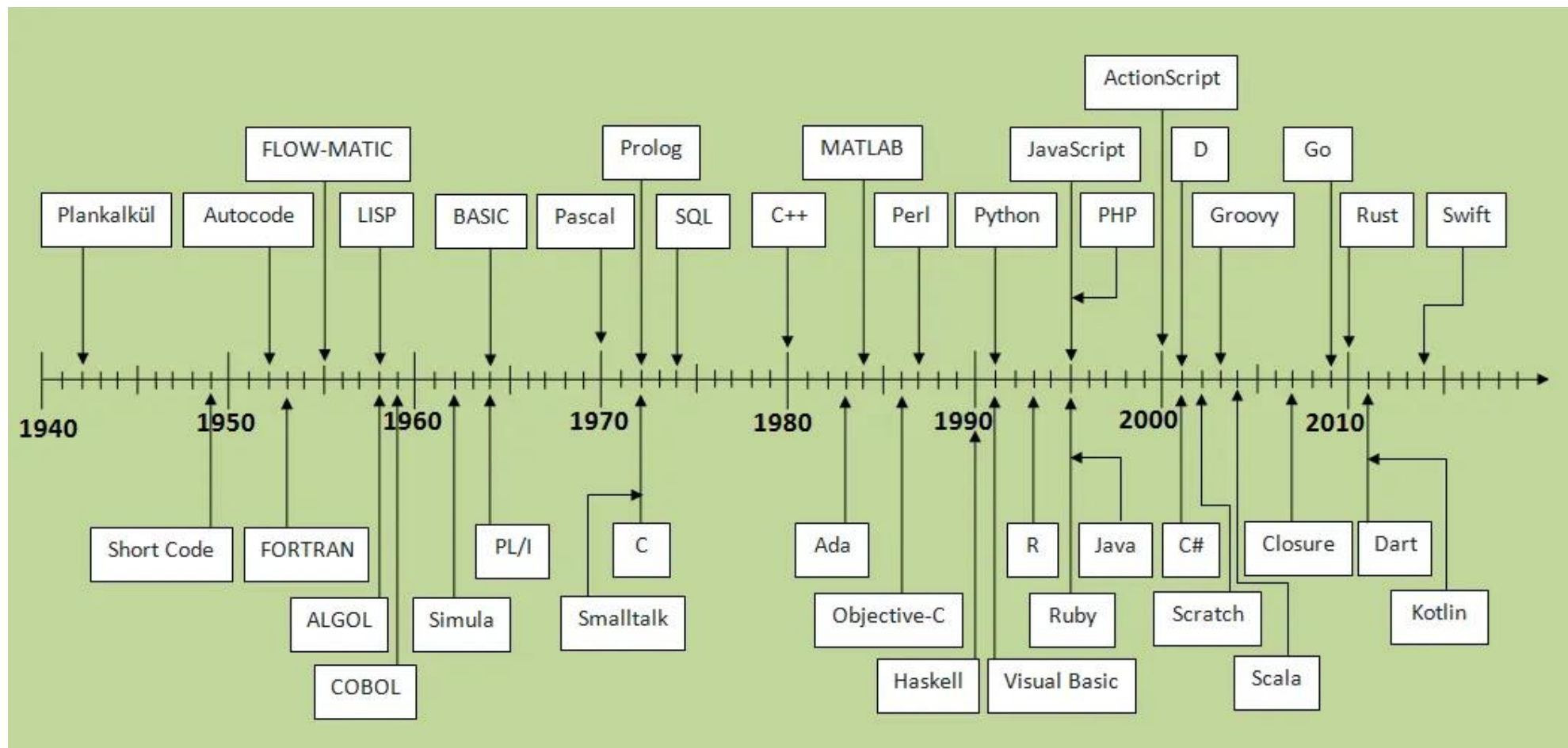
Стандарт/Документация

- PEP-8 - <https://www.python.org/dev/peps/pep-0008/>
- <https://www.python.org/>
- <https://github.com/python/cpython>

Экспертный уровень

- Лучано Рамальо: Python. К вершинам мастерства
- Mitchell L. Model. Bioinformatics Programming Using Python // O'Reilly 2010.

Языки программирования(ЯП)



И это далеко не все...

Интерпретируемые ЯП

- + Кроссплатформенность, пошаговое отслеживание выполнения программы, модификация программы во время исполнения, меньшие затраты времени на разработку и отладку, простой способ создания переносимых программ, не требует затрат на компиляцию небольших программ ...
- Основным недостатком является более медленное выполнение программ. Необходимость везде и всегда таскать с собой рантайм(среду выполнения кода)*.

* Последнее замечание конечно нивелируется тем, что Python весьма популярен и стандартная библиотека как и сам интерпретатор поставляется обычно со всеми более или менее популярными дистрибутивами ОС “из под капота”.

Интерпретируемые ЯП

Интерпретируемые

- Python
- Java
- Bash
- Ruby
- Perl
- C#
- ...

Компилируемые

- C
- C++
- Assembler
- Go
- Pascal
- Rust
- Swift
- ...

Почему Python?

- Постоянно растущая популярность языка дает импульс для развития и совершенствования
- Большое комьюнити
- Огромное многообразие библиотек для различных нужд от программирования микроконтроллеров, до машинного обучения
- Машинное обучение
- Низкий порог вхождения
 - прост в освоение
 - можно быстро начать писать код для продакшена
- Универсальность

Рейтинги

Aug 2021	Aug 2020	Change	Programming Language	Ratings	Change
1	1		 C	12.57%	-4.41%
2	3	▲	 Python	11.86%	+2.17%
3	2	▼	 Java	10.43%	-4.00%
4	4		 C++	7.36%	+0.52%

<https://www.tiobe.com/tiobe-index/>

PULL REQUEST

Year: 2021 | Quarter: 2

# Ranking	Programming Language	Percentage (YoY Change)	YoY Trend
1	JavaScript	19.014% (+0.225%)	
2	Python	16.351% (+0.243%)	
3	Java	12.817% (+2.086%)	
4	Go	7.574% (-1.348%)	

https://madnight.github.io/githut/#/pull_requests/2021/2

Community

Repositories	2M+
Code	132M
Commits	28M+
Issues	6M
Discussions	16K
Packages	5K
Marketplace	172
Topics	4K
Wikis	280K
Users	129K



Python

Python is a dynamically typed programming language.

[See topic](#)

☆ Star

Showing 2,170,439 available repository results ?

Sort: Best match ▾



TheAlgorithms/Python

♡ Sponsor

All Algorithms implemented in **Python**

python

hacktoberfest

education

algorithm

practice

interview

sorting-algorithms

learn

algos

algorithm-competitions

sorts

algorithms-implemented

community-driven

searches

☆ 118k

● Python

MIT license

Updated 11 hours ago

1 issue needs help

Порог вхождения

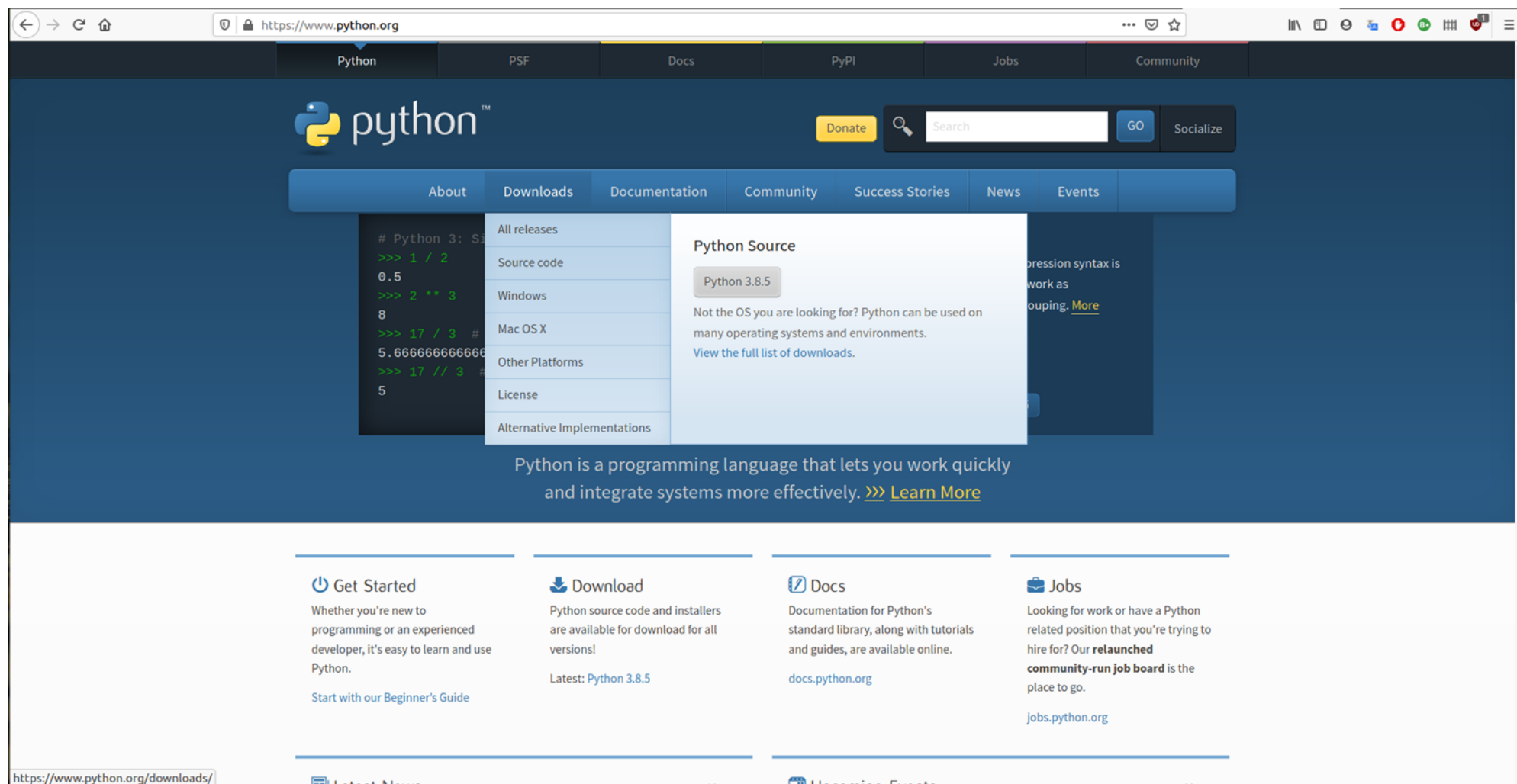
Начать писать на Python при понимании основных конструкций очень просто \Rightarrow Можно быстро и, следовательно, дешево, реализовать программное решение

Универсальность

Python - подходит для широкого круга задач, как задачи, связанные с машинным обучением, так и написание высоконагруженных сервисов.

- Instagram
- Youtube
- Google
- Facebook
-

Установка



<https://python.org>

Версии Python

- Python 2 вышел 2010 году последняя версия 2.7.16 - исправлялись только баги с января 2020 года поддержка прекращена.
- Python 3 появился в 2008, является актуальной версией языка.
Текущая стабильная версия 3.10
Python 3 не гарантирует совместимости кода с Python 2

НАСТОЯТЕЛЬНО РЕКОМЕНДУЮ РАБОТАТЬ С PYTHON >= 3.x

Что входит в стандартную библиотеку

Стандартная библиотека предоставляет большой список функций и типов данных, которыми можно оперировать не устанавливая при этом сторонние библиотеки. Полный список можно найти перейдя по ссылке <https://docs.python.org/3/library/>.

Введение

- запуск интерактивной строки интерпретатора
 - для Linux | MacOS в командной строке запуск `python` | `python3`
 - Windows обычно после установки `python` добавляет в системную переменную `PATH`

```
[22:36:03] serg :: serg-pc → ~»  
python3  
Python 3.8.2 (default, Jul 16 2020, 14:00:26)  
[GCC 9.3.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> █
```


Использование интерактивной строки интерпретатора

```
python
Python 3.8.10 (default, Jun 2 2021, 10:49:15)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 1 + 1
2
>>> 10 + 1 - (7 - 2) * 3
-4
>>> a = 1
>>> b = 2
>>> c = a + b
3
>>>
```

Версия Python

- Python 3.8.10 (2021-06-02) - это последняя версия Python 3.8.10.
- Python 3.8.10 (2021-06-02) - это последняя версия Python 3.8.10.
- Python 3.8.10 (2021-06-02) - это последняя версия Python 3.8.10.
- Python 3.8.10 (2021-06-02) - это последняя версия Python 3.8.10.
- Python 3.8.10 (2021-06-02) - это последняя версия Python 3.8.10.

Использова

Среды разработки (IDE)

- Visual Studio Code <https://code.visualstudio.com/> - бесплатная IDE, мультязычная, много плагинов все бесплатные
- PyCharm - <https://www.jetbrains.com/pycharm/> - условно бесплатная для студентов доступна enterprise версия
- Spyder - <https://www.spyder-ide.org/> - бесплатная IDE
- Любой текстовый редактор (vim, nano, atom, notepad++...)

Комментарии в коде (как оформлять?)

- Что такое комментарии?
- Поддерживает ли этот функционал Python?
- Как обрабатывает комментарии интерпретатор?
- Как их оформлять?
- Я хочу много комментариев в своем супер приложении, как мне это сделать?

Примеры комментариев (однострочный)

Однострочные комментарии начинаются с символа '#'. Если интерпретатор встречает подобный символ, то игнорирует все что написано после него

```
# this is one line comment it starts with symbol '#'
```

Примеры комментариев (многострочный)

Многострочные комментарии можно оформлять по разному

1. Как группу однострочных комментариев, написанных один под другим
2. Через три подряд идущих открывающих/закрывающих символа кавычки “ или ’

```
# Multi line comment  
# line 1  
# line 2
```

```
""" Other multiline comment  
line 1  
line 2  
"""
```

Комментарии и документация

Документация - подробная или не очень описание функционала вашего кода...


Комментарии тесно связаны с документацией. Документация для модуля помещается в начало файла и обычно оформляется как многострочный комментарий начинающийся `"""`. Для функции документация также помещается перед определением тела функции

```
In [1]: def foo(arg1, arg2):  
...:     """ This function literally do nothing  
...:     :param arg1: description of arg1  
...:     :param arg2: description of arg2  
...:     """  
...:     pass  
...:
```

Функция help

После оформления комментария документации у пользователя появляется возможность посмотреть документацию с помощью встроенной функции help

```
help(foo)
```



```
Help on function foo in module __main__:  
  
foo(arg1, arg2)  
    This function literally do nothing  
  
    :param arg1: description of arg1  
    :param arg2: description of arg2  
(END)
```

Базовые типы данных

Числовые типы данных - int, float complex

- **int** (integer) - целое число, например 10, 1, 0,
- **float** - числа с плавающей точкой например 1.2, 3.14,
- **complex** - комплексные числа определяются двумя числами вещественной частью и мнимой:

$$\text{num} = a + i*b \rightarrow \text{где } i = \sqrt{-1}$$

Логические типы

- True/False

Итераторы

- Специальные объекты позволяющие пройти по последовательности

Базовые типы данных

Последовательности

- list -> [1, 3, 4, 5]
- tuple -> (1, 2, 3, 4)
- range -> range(start, end, step) e.g. range(0,10,1)

Текстовые последовательности

- str -> "Hello Python!"

Пустое значение

- None

Словари/множества

- set -> set(1,2,3,4)
- frozenset -> frozenset(1,2,3,4)
- dict -> {"John Doe": "+7903222334", "Albert Einstein": "+142345553", ...}

None

В Python None - это константа, которая служит для идентификации того, что переменная которая ссылается на None в данный момент не указывает ни на какой объект в оперативной памяти.

Аналоги для других ЯП: NULL, null, nil, ...

Часто используется, для инициации переменных значение которых еще не вычислено и будет вычислено позже. **Обращение к таким переменным как к объекту содержащему полезную информацию, без проверки его на пустоту приведет к ошибке времени выполнения!**

Последовательности (Список)

Последовательность элементов, сохраняющую порядок. То есть элементы попадающие в список будут находиться в нем в том порядке, в котором были добавлены. Списки поддерживают операцию индексации [index]

```
In [1]: l = [1,2,3]
```

```
In [2]: l
```

```
Out[2]: [1, 2, 3]
```

```
In [4]: l = list()
```

```
In [5]: l.append(1)
```

```
In [6]: l.append(2)
```

```
In [7]: l.append(3)
```

```
In [8]: l
```

```
Out[8]: [1, 2, 3]
```

Кортеж (Tuple)

Также последовательность элементов, сохраняющая порядок следования. **НО** эта последовательность. То есть после определения кортежа вы не сможете поменять его состав или увеличить/уменьшить... Хотя операцию индексации этот тип данных также поддерживает.

```
In [10]: t = (1,2,3)
```

```
In [11]: t  
Out[11]: (1, 2, 3)
```

```
In [16]: t[1]  
Out[16]: 2
```

```
In [14]: t = tuple([1,2,3])
```

```
In [15]: t  
Out[15]: (1, 2, 3)
```

```
In [17]: t[1] = 4
```

```
TypeError
```

```
<ipython-input-17-87b0f225887f> in <module>
```

```
----> 1 t[1] = 4
```

```
TypeError: 'tuple' object does not support item assignment
```

```
Traceback (most recent call last)
```

Строки

Строка - буквенно знаковая последовательность символов

Кодировка (UTF-8) - наиболее распространенная кодировка, использует для кодирования символов от 1 до 4 байтов (1 байт - 8 бит)

```
In [3]: s_example1 = "String example one"
In [4]: s_example2 = "String example two"
```

```
In [1]: привет="Hello"
In [2]: print(привет)
Hello
```

Итераторы

Специализированные объекты позволяющие итерироваться(“пробежаться”) по коллекции элементов, при этом каждый следующий элемент может генерироваться на каждой итерации.

```
In [23]: g = (i for i in range(10))

In [24]: for e in g:
...:     print(e)
...:

0
1
2
3
4
5
6
7
8
9
```

Словари

Структура хранящая элементы как ключ → значение. НЕ СОХРАНЯЕТ ПОРЯДОК ЭЛЕМЕНТОВ. Обеспечивает быстрой доступ к элементам, быструю вставку/удаление

```
In [27]: d = dict(name='Sergey', surname='khayrulin')  
  
In [28]: d  
Out[28]: {'name': 'Sergey', 'surname': 'khayrulin'}  
  
In [29]: d = {'name': 'Sergey', 'surname': 'Khayrulin'}  
  
In [30]: d  
Out[30]: {'name': 'Sergey', 'surname': 'Khayrulin'}
```

Множества

Dict-like объект, отличия от словаря заключается в том, что множества хранят только ключи без значений, кроме того сущность множество предоставляет набор основных теоретико-множественных операций - объединения, пересечения, дополнение и так далее...

```
In [32]: s = {1,2,3}
```

```
In [33]: s
```

```
Out[33]: {1, 2, 3}
```

```
In [34]: s = set([1,2,3])
```

```
In [35]: s
```

```
Out[35]: {1, 2, 3}
```


Базовые типы данных

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

Арифметические операции над числами

Operation	Result	Notes	Full documentation
<code>x + y</code>	sum of <code>x</code> and <code>y</code>		
<code>x - y</code>	difference of <code>x</code> and <code>y</code>		
<code>x * y</code>	product of <code>x</code> and <code>y</code>		
<code>x / y</code>	quotient of <code>x</code> and <code>y</code>		
<code>x // y</code>	floored quotient of <code>x</code> and <code>y</code>	(1)	
<code>x % y</code>	remainder of <code>x / y</code>	(2)	
<code>-x</code>	<code>x</code> negated		
<code>+x</code>	<code>x</code> unchanged		
<code>abs(x)</code>	absolute value or magnitude of <code>x</code>		abs()
<code>int(x)</code>	<code>x</code> converted to integer	(3)(6)	int()
<code>float(x)</code>	<code>x</code> converted to floating point	(4)(6)	float()
<code>complex(re, im)</code>	a complex number with real part <code>re</code> , imaginary part <code>im</code> . <code>im</code> defaults to zero.	(6)	complex()
<code>c.conjugate()</code>	conjugate of the complex number <code>c</code>		
<code>divmod(x, y)</code>	the pair <code>(x // y, x % y)</code>	(2)	divmod()
<code>pow(x, y)</code>	<code>x</code> to the power <code>y</code>	(5)	pow()
<code>x ** y</code>	<code>x</code> to the power <code>y</code>	(5)	

Приведение типов

Явное - из название должно быть понятно, что подобное приведение делается явно вами как программистами

Неявное - скорее всего за вас это делает интерпретатор. Вы можете не подозревать о нем, что может приводить к обидным ошибкам, которые еще и искать трудно :(

Явные операции приведения типа (Python)

- `int` - приведение к целому числу
- `float` - приведение к числу с плавающей точкой
- `str` - приведение к строке
- `bool` - приведение к логическому типу

bool()

Стоит **ЗНАТЬ**, что любое значение не равное 0 при приведении типа к bool результатом будет **True**, соответственно для пустых значение таких как 0, ""(пустая строка), [](пустой список) ... результатом приведения к bool будет **False**

```
In [17]: bool(1)
Out[17]: True

In [18]: bool(0)
Out[18]: False

In [19]: bool("")
Out[19]: False

In [20]: bool([])
Out[20]: False
```

Переменные - именования

Полный список требований лучше почитать [здесь](#). Здесь кратко:

1. Имя переменной содержит только цифробуквенные символы и символ `_`. **НАРУШЕНИЕ ЭТОГО ПРАВИЛА ВЛЕЧЕТ ОШИБКУ ИНТЕРПРЕТАЦИИ!**
2. Имя переменной **НЕ МОЖЕТ** начинаться с цифры. **ТОЖЕ ЧРЕВАТО ОШИБКАМИ!**
3. Длина имени > 1 (это не строгое требование в некоторых случаях вполне оправдано в основном историческими причинами)
4. Максимальное имя тоже ограничено
5. Имя переменной должно быть лаконичным и отражать смысл данных, которые вы предполагаете получать при работе с этой переменной

Переменные - именования (стиль)

Вообще в Python принято пользоваться двумя стилями для именования сущностей программы

CamelCase - для именования имен классов и комплексных сущностей (для разделения смысловых частей названия используется заглавная буква)

пример -> MyClass, HTTPServer,...

underscore - для именования всего остального (для разделения смысловых частей названия используется символ '_')

пример -> my_function(...), my_variables, ...

Переменные - именования

Плохое имя переменной, но интерпретатор молчаливо пережует его

f, aa, d1, r55, m32, this_is_very_long_name_for_variables...

Примеры имен, которые вызовут ошибку интерпретации

1, 1f, aaa@aaa ...

Хорошее имя переменной

some_text, db_connection, received_data, point, user_list,...

Переменные инициализация (синтаксис)

Переменные инициализируются через оператор присваивания =

Пример: `variable_name = variable_value`

Объявление переменных в Python

```
# объявление переменной
```

```
x = 10
```

```
# Переопределение значения переменной,
```

```
# то есть после этой операции переменная
```

```
# будет указывать на другой объект в памяти.
```

```
some_text = 'Hello World'
```

```
pi = 3.14
```

```
# Изменение переменной. После этой операции переменная
```

```
# x указывает на новое значение 4.14
```

```
x = x + 1
```

Динамическая типизация

***Strong** typing means that the type of a value doesn't change in unexpected ways. A string containing only digits doesn't magically become a number, as may happen in Perl. Every change of type requires an explicit conversion.*

***Dynamic** typing means that runtime objects (values) have a type, as opposed to static typing where variables have a type.*

Python - **динамически** типизированный язык.

При динамической типизации переменная не знает тип значения на которое в данный момент эта переменная указывает (**ЭТО ВАЖНО**) - при этом само значение хранит эту информацию.

Переменные(базовые операции)

Для сокращения записи, если выражение

подразумевает изменение той же переменной,

то разумно использовать следующие варианты записи

`x += 1` *# тоже самое что и $x = x + 1$*

`x -= 1` *# тоже самое что и $x = x - 1$*

`x *= 1` *# тоже самой что и $x = x * 1$*

`x /= 2` *# тоже самое что и $x = x / 2$*

`x **= 2` *# тоже самое что и $x = x ** 2$*

Присвоение

`x = 1`

`y = 2`

`x = y` *# теперь переменная x равна 2*

`z = x + y` *# теперь переменная z равна 4*

Работа с вводом/выводом

Взаимодействие с пользователем может настраиваться через стандартные потоки ввода и вывода `stdin/stdout`.

Для получения входной информации от пользователя можно воспользоваться вызовом функции [`input\(...\)`](#)

Для вывода информации из программы можно воспользоваться функцией [`print\(...\)`](#)

Важно: функция `input` блокирует выполнение скрипта и ждет ввода пользователя, для того чтобы продолжить работу. Кроме того сама функция возвращает введенную строку, **НЕ ЗАБЫВАЙТЕ КОНВЕРТИРОВАТЬ ТИПЫ**

Работа с вводом/выводом

```
In [21]: my_name = input("Put your name here: ")  
Put your name here: Sergey
```

```
In [22]: print(my_name)  
Sergey
```

```
In [23]: █
```

Домашнее задание

1. Установить Python, в командной строке “import this”, установить IDE
2. Разработайте приложение принимающее на вход два числа и выводящее сумму этих чисел
3. Реализовать программу, которая спрашивает у пользователя: имя, фамилию, год рождения. После ввода всех данных программа должна выводить строку следующего вида:

```
"Hello {Name} {Surname} your age is {year} year"
```