Введение в программирование на Python 3

История возникновения языка Python 3

Python был разработан в конце 1989 г. Гуидо ван Россумом во время рождественских каникул, когда его исследовательская лаборатория была закрыта и ему просто некуда было деваться. Он позаимствовал многие средства программирования, присущие другим языкам.

Гуидо обожал телевизионную передачу “Летающий цирк питона Монти”, и когда пришло время дать название своему языку, он выбрал имя Python. В 1991 г. после испытаний и экспериментов в узком кругу друзей и коллег по работе, Python был размещен для общего доступа на суд широкой общественности.

История возникновения языка Python 3

В отличие от других языков программирования, Python не только распространяется совершенно бесплатно, он не имеет абсолютно никаких ограничений в условиях применения. Никто не ограничивает коммерческое использование программных продуктов, написанных на этом языке, без каких-либо лицензионных отчислений.

Программисты также вольны модернизировать язык, не ставя в известность автора.

Наличие дружелюбного, отзывчивого сообщества пользователей считается наряду с дизайнерской интуицией Гвидо одним из факторов успеха Python. Развитие языка происходит согласно чётко регламентированному процессу создания, обсуждения, отбора и реализации документов PEP (Python Enhancement Proposal) — предложений по развитию Python.

3 декабря 2008 г., после длительного тестирования, вышла первая версия Python 3000 (или Python 3.0, также используется сокращение Py3k). В Python 3000 устранены многие недостатки архитектуры с максимально возможным (но не полным) сохранением совместимости со старыми версиями Питона. На сегодня поддерживаются обе ветви развития (Python 3.0 и 2.x).

Философия Python 3

Разработчики языка Python придерживаются определённой философии программирования, называемой "Дзэном Питона". Ее автором считается Тим Пейтерс

Текст доступен по команде **import this**.

Философия Python 3

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.
- Читабельность имеет значение.
- Особые случаи не настолько особые, чтобы нарушать правила.
- Хотя практичность побеждает стремление к чистоте.
- Ошибки никогда не должны замалчиваться.

Философия Python 3

Текст философии:

- Если не замалчиваются явно.
- Если видишь 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, черт возьми, — отличная штука! Будем делать их побольше!

Для чего используется Python 3

Python может использоваться для многих целей, например:

- Разработка прикладного ПО
- Разработка мобильных приложений
- Разработка web-приложений
- В качестве встраиваемого скриптового языка во многих играх, и программах
- В научных рассчетах

Hello, World!

```
>>> print ('Hello, World!')
'Hello, World!'
```

Типизация в Python

Python – язык со строгой динамической типизацией.

Т.е. тип используемой переменной трактуется в зависимости от значения, при этом не допускается неявное преобразование (например, сложение строки и числа).

Типы данных в Python 3

В Python имеется множество встроенных типов данных. Наиболее важные из них:

- **Логический**
- **Числа**: целые, с плавающей точкой, дробные и комплексные
- **Строки** — последовательности символов Юникода
- **Байты** и **массивы байтов**
- **Списки** — упорядоченные последовательности значений
- **Кортежи** — упорядоченные неизменяемые последовательности значений
- **Множества** — неупорядоченные наборы значений
- **Словари** — неупорядоченные наборы пар вида ключ-значение

Типы данных в Python 3. Логический.

Логический тип данных может принимать одно из двух значений: истина или ложь. В Python имеются две константы с понятными именами True (от англ. true — истина) и False (от англ. false — ложь), которые можно использовать для непосредственного присвоения логических значений.

Из-за некоторых обстоятельств, связанных с наследием, оставшимся от Python 2, логические значения могут трактоваться как числа. True как 1, и False как 0.

```
>>> size = 1
>>> size < 0
False
>>> size = 0
>>> size < 0
False
>>> size = -1
>>> size < 0
True
```

```
>>> True + True
2
>>> True - False
1
>>> True * False
0
>>> True / False
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: int division or modulo by zero
```

Типы данных в Python 3. Числа.

Python поддерживает как целые числа, так и с плавающей точкой. И нет необходимости объявлять тип для их различия - Python определяет его по наличию или отсутствию десятичной точки.

```
>>> type(1)                    ①
<class 'int'>
>>> isinstance(1, int)         ②
True
>>> 1 + 1.0                    ③
2.0
>>> type(2.0)
<class 'float'>
```

① Можно использовать функцию type() для проверки типа любого значения или переменной.

② Функцию isinstance() тоже можно использовать для проверки принадлежности значения или переменной определенному типу.

③ Сложение значений типа int и float дает в результате float. Для выполнения операции сложения Python преобразует значение типа int в значение типа float, и в результате возвращает float.

Типы данных в Python 3. Строки.

Базовые операции:

Операция	Пример кода

Конкатенация (сложение)	>>> S1 = 'spam'
	>>> S2 = 'eggs'
	>>> print(S1 + S2)
	'spameggs'

Дублирование строки	>>> print('spam' * 3) spamspamspam
Длина строки	>>> len('spam') 4

Доступ по индексу	>>> S = 'spam'
	>>> S[0]
	's'

Извлечение среза	>>> s = 'spameggs'
	>>> s[3:5]
	'me'

Типы данных в Python 3. Байты.

Байт - минимальная единица хранения и обработки цифровой информации.

Последовательность байт представляет собой какую-либо информацию (текст, картинку, мелодию...).

Пример с байтовой строкой:

```
>>> b'bytes'
b'bytes'
>>> 'Байты'.encode('utf-8')
b'\xd0\x91\xd0\xb0\xd0\xb9\xd1\x82\xd1\x8b'
>>> bytes('bytes', encoding = 'utf-8')
b'bytes'
>>> bytes([50, 100, 76, 72, 41])
b'2dLH)‘
>>> b'\xd0\x91\xd0\xb0\xd0\xb9\xd1\x82\xd1\x8b'.decode('utf-8')
'Байты'
```

Типы данных в Python 3. Массивы байтов.

Bytearray - массив байт.

От типа **bytes** отличается только тем, что является изменяемым.

```
>>> b = bytearray(b'hello world!')
>>> b
bytearray(b'hello world!')
>>> b[0]
104
>>> b[0] = 105
>>> b
bytearray(b'iello world!')
```

Типы данных в Python 3. Списки.

В Python список — это нечто подобное Java-классу ArrayList, который может хранить произвольные объекты и динамически расширяться по мере добавления новых элементов.

```
>>> a_list = ['a', 'b', ‘c', ‘d', ‘e']                  # Объявление списка
>>> a_list[1:3]                                         # Срез со второго по третий символ
['b', ‘c']
>>> a_list += [‘f’]                                     # Добавление элемента в список
['a', 'b', ‘c', ‘d', ‘e‘, ‘f’]
>>> a_list.append(‘g’)                                  # Еще один вариант добавления элементов
['a', 'b', ‘c', ‘d', ‘e‘, ‘f’, ‘g’]
>>> a_list.extend([‘h’, ‘I’])                           # И еще один
['a', 'b', ‘c', ‘d', ‘e‘, ‘f’, ‘g’, ‘h’, ‘I’]
>>> a_list.insert(0, ‘a’)                               # Добавление элемента в указанную позицию
['a', 'a', 'b', ‘c', ‘d', ‘e‘, ‘f’, ‘g’, ‘h’, ‘I’]
>>> a_list.count(‘a’)                                   # Индексы вхождений элемента в список
0, 1
>>> a_list.index(‘a’)                                   #Индекс первого вхождения в список
0
```

Типы данных в Python 3. Списки.

В Python список — это нечто подобное Java-классу ArrayList, который может хранить произвольные объекты и динамически расширяться по мере добавления новых элементов.

```
>>> ‘j’ in a_list
False
>>> len(a_list)
10
>>> a_list.remove(‘a’)
['a', 'b', ‘c', ‘d', ‘e‘, ‘f’, ‘g’, ‘h’, ‘I’]
>>> a_list.pop([1])
‘b’
>>> a_list.reverse()
[‘i’, ‘h’, ‘g’, ‘f’, ‘e’, ‘d’, ‘c’, ‘b’, ‘a’]
>>> a_list.copy()
[‘i’, ‘h’, ‘g’, ‘f’, ‘e’, ‘d’, ‘c’, ‘b’, ‘a’]
>>> a_list.clear()
[]
```

```
# Проверка на вхождение элемента в список
# Длинна списка

# Удаление элемента из списка

# Выдергиваем элемент из списка

# Отзеркаливание списка

# Копирование списка

# Очищение списка
```

Типы данных в Python 3. Кортежи.

Кортеж - неизменяемый список.

Зачем они нужны:

- Защита от дурака (неизменяемы)
- Меньший размер в памяти
- Можно использовать в качестве ключей словаря
- Присваивание значений нескольким переменным

У кортежей отсутствуют методы.

Типы данных в Python 3. Множества.

Множество — это неупорядоченная коллекция без дублирующихся элементов.

Основные способы использования — проверка на вхождение и устранение дублирующихся элементов. Объекты этого типа поддерживают обычные математические операции над множествами, такие как объединение, пересечение, разность и симметрическая разность.

Для создания пустого множество следует использовать set().

Типы данных в Python 3. Множества.

Методы и операции	Значение
A | B A.union(B)	Возвращает множество, являющееся объединением множеств A и B.
A |= B A.update(B)	Добавляет в множество A все элементы из множества B.
A & B A.intersection(B)	Возвращает множество, являющееся пересечением множеств A и B.
A &= B A.intersection_update(B)	Оставляет в множестве A только те элементы, которые есть в множестве B.
A - B A.difference(B)	Возвращает разность множеств A и B (элементы, входящие в A, но не входящие в B).
A -= B A.difference_update(B)	Удаляет из множества A все элементы, входящие в B.

Типы данных в Python 3. Множества.

Методы и операции	Значение
A ^ B A.symmetric_difference(B)	Возвращает симметрическую разность множеств A и B (элементы, входящие в A или в B, но не в оба из них одновременно).
A ^= B A.symmetric_difference_update(B)	Записывает в A симметрическую разность множеств A и B.
A <= B A.issubset(B)	Возвращает true, если A является подмножеством B.
A >= B A.issuperset(B)	Возвращает true, если B является подмножеством A.
A < B	Эквивалентно A <= B and A != B
A > B	Эквивалентно A >= B and A != B

Типы данных в Python 3. Множества.

```
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a
{'a', 'r', 'b', 'c', 'd'}
>>> a - b
{'r', 'd', 'b'}
>>> a | b
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b
{'a', 'c'}
>>> a ^ b
{'r', 'd', 'b', 'm', 'z', 'l'}
```

```
# уникальные буквы в a

# буквы в a, но не в b

# все буквы, которые встречаются в a или в b
# буквы, которые есть и в a и в b

# буквы в a или в b, но не в обоих
```

Типы данных в Python 3. Словари.

Словарь (dictionary) — это ассоциативный массив или хеш. Это неупорядоченное множество пар ключ: значение с требованием уникальности ключей. Пара фигурных скобок {} создает пустой словарь.

В отличие от последовательностей, доступ к элементам словаря производится по ключу, а не по индексу, ключ может быть любого типа, ключ не допускает изменений.

Типы данных в Python 3. Словари.

Метод	Значение
dict()	создание словаря
len()	возвращает число пар
clear()	удаляет все значения из словаря
copy()	создает псевдокопию словаря
deepcopy()	создает полную копию словаря
fromkeys()	создание словаря
get()	получить значение по ключу
has_key()	проверка значения по ключу
items()	возвращает список значений
iteriyems()	возвращает итератор
keys()	возвращает список ключей
iterkeys()	возвращает итератор ключей
pop()	извлекает значение по ключу

Типы данных в Python 3. Словари.

Метод	Значение
popitem()	извлекает произвольное значение
update()	изменяет словарь
values()	возвращает список значений
itervalues()	возвращает итератор на список значений
del()	оператор, удаляет пару по ключу

Цикл While

While - один из самых универсальных циклов в Python, поэтому довольно медленный. Выполняет тело цикла до тех пор, пока условие истинно.

```
>>> i = 5
>>> while i < 15:
...     print(i)
...     i = i + 2
...
5
7
9
11
13
```

Цикл For

Цикл **for** немного сложнее и менее универсальный, но выполняется гораздо быстрее цикла **while**. Этот цикл проходится по любому итерируемому объекту (например строке или списку), и во время каждого прохода выполняет тело цикла.

```
>>> for i in 'hello world':
...     print(i * 2, end='')
...
hheelllloo  wwoorrlldd
```

Оператор continue

Оператор continue начинает следующий проход цикла, минуя оставшееся тело цикла (for или while).

```
>>> for i in 'hello world':
...     if i == 'o':
...         continue
...     print(i * 2, end='')
...
hheellll  wwrrlldd
```

Else

Оператор break досрочно прерывает цикл.

```
>>> for i in 'hello world':
...     if i == 'a':
...         break
... else:
...     print('Буквы a в строке нет')
...
Буквы a в строке нет
```

Оператор break

Оператор break досрочно прерывает цикл.

```
>>> for i in 'hello world':
...     if i == 'o':
...         break
...     print(i * 2, end='')
...
hheellll
```