БИБЛИОТЕКИ ЯЗЫКА PYTHON ДЛЯ КОМПЬЮТЕРНЫХ ВЫЧИСЛЕНИЙ И МОДЕЛИРОВАНИЯ

Основные особенности языка Python

Лобанов Алексей Владимирович
Главный специалист отдела разработки и внедрения АИС,
Ассистент кафедры информационных компьютерных технологий РХТУ им. Д.И. Менделеева

ТЕМЫ

- Ключевые особенности языка **Python**
- Области применения языка
- Популярные среды для разработки на языке Python
- Основные типы переменных
- Простейшие операции над переменными
- Логические операции
- Конструкции **if, else, elif**
- Списки
- Циклы **while** и **for**

ПОЛЕЗНЫЕ РЕСУРСЫ

- https://www.python.org/
- https://docs.python.org/
- https://pythonworld.ru/
- https://www.anaconda.com/
- https://jupyterlab.readthedocs.io/
- https://www.spyder-ide.org/
- https://www.codewars.com/

КЛЮЧЕВЫЕ ОСОБЕННОСТИ ЯЗЫКА PYTHON

Python – это высокоуровневый кроссплатформенный интерпретируемый язык программирования общего назначения, появившийся порядка 30 лет тому назад. Он ориентирован на повышение эффективности разработчика и читаемости кода. Синтаксис у данного языка минималистичен, и сосредоточен на сокращении программного кода при помощи коротких языковых конструкций, а стандартная библиотека включает большой объем поддерживаемых функций.

Python способен поддерживать множество парадигм программирования, включая объектно-ориентированное, императивное, структурное и функциональное программирование. Если говорить об основных чертах языка, то следует отметить динамическую типизацию, автоматическое управление памятью, механизм обработки исключений, а также полную интроспекцию. Помимо этого, Python поддерживает многопоточность и способность работать с высокоуровневыми структурами данных. Кроме того, в Python имеется возможность разбивать программы на отдельные модули, их же в свою очередь в дальнейшем можно объединять в пакеты.

КЛЮЧЕВЫЕ ОСОБЕННОСТИ ЯЗЫКА PYTHON

Преимущества:

- Качество программного обеспечения
- Высокая скорость разработки
- Динамическая типизация
- Переносимость программ
- Поддержка библиотек

Недостатки:

- Необходимость установки интерпретатора
- Относительно низкая скорость выполнения программ
- Сложности с динамической типизацией

КЛЮЧЕВЫЕ ОСОБЕННОСТИ ЯЗЫКА PYTHON

Python активно развивается и является одним из самых прогрессивных языков программирования. Новые версии выходят с частотой примерно раз в два с половиной года. (текущая версия 3.10.7) Для сравнения, Python является 4-м по популярности на портале StackOverflow, который создан для обсуждения вопросов, касающихся программирования. На данном ресурсе задано уже более 500 тысяч вопросов по данному языку. Кроме того, этот язык является 4-м по использованию на крупнейшем портале для хранения репозиториев GitHub. Такая популярность в первую очередь обусловлена широкими возможностями, которые предоставляет Python своим разработчикам.

Python является одним из самых быстрорастущих языков программирования и поданным аналитики это может быть обусловлено многими факторами. Однако, основополагающий фактор – это его высокая применимость в науке, благодаря обширному числу библиотек по данной тематике, а также низкому порогу вхождения в данный язык.

ОБЛАСТИ ПРИМЕНЕНИЯ ЯЗЫКА

ПОПУЛЯРНЫЕ СРЕДЫ ДЛЯ РАЗРАБОТКИ НА ЯЗЫКЕ PYTHON

Поддерживают **Python**

Visual Studio Code

Eclipse

Созданы для **Python**

PyCharm

Anaconda

Онлайн работа с **Python**

Yandex Cloud Functions

Google Collab

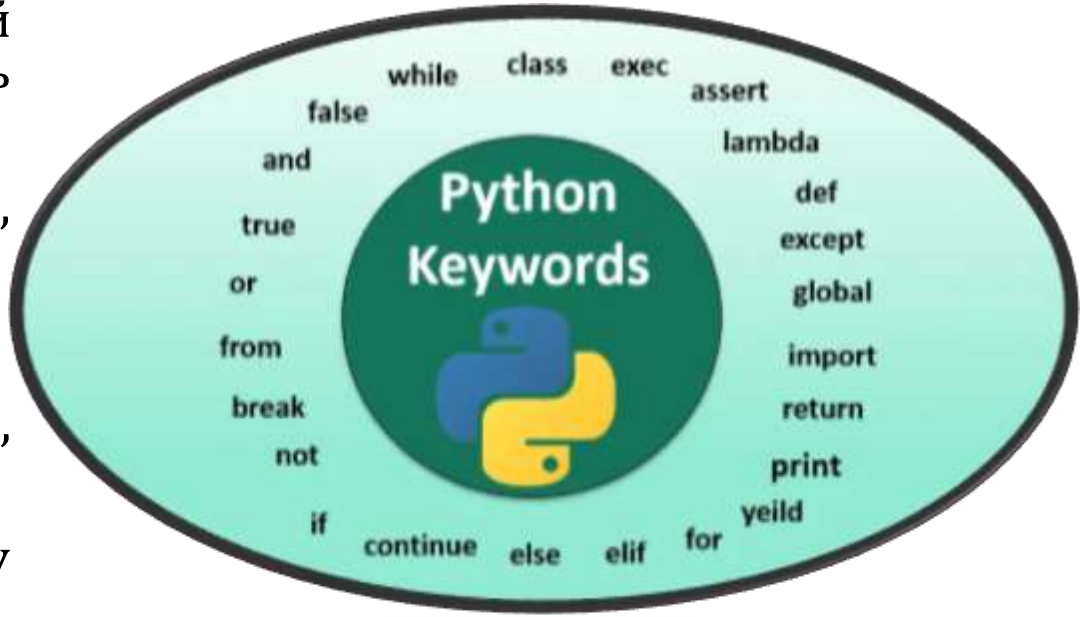
ОСНОВНЫЕ ТИПЫ ПЕРЕМЕННЫХ

Название	Тип	Пример и краткое описание
Integer	int	3, 79, 0, -35 - целые числа
Floating point	float	7.98, -0.905, 1.4 - упорядоченная последовательность символов
String	str	"hello", 'Ivan', "2345" - упорядоченная последовательность символов
List	list	["Hello", 55, 26.6, "Hello"] - упорядоченная последовательность объектов
Dictionary	dict	{“name”: “Ivan”, “age”: 20} - упорядоченная последовательность пар ключ-значение
Tuple	tup	("Hello", 55, 26.6, "Hello") - упорядоченная неизменяемая последовательность объектов
Boolean	bool	True, False - логический тип данных

ОСОБЕННОСТИ ИМЕНОВАНИЯ

Переменные в Python не требуют объявления типа переменной (так как Python – язык с динамической типизацией) и являются ссылками на область памяти. Правила именования переменных:

- Имя переменной может состоять только из букв, цифр и знака подчёркивания;
- Имя не может начинаться с цифры;
- Имя не может содержать специальных символов @, $, %;
- Имя переменной не должно равняться ключевому слову;
- Желательно, чтобы имя было «говорящим»



ОСОБЕННОСТИ ИМЕНОВАНИЯ

- Если название состоит из нескольких слов, то они разделяются нижним подчеркиванием
- Избегайте написания русских слов латиницей (вместо znachenie используйте value)
- Все вышеперечисленное может не соблюдаться согласно устоявшимся названиям из научной области решаемой задачи (например G - для расхода пара или R - для универсальной газовой постоянной)

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

ПРОСТЕЙШИЕ ОПЕРАЦИИ НАД ПЕРЕМЕННЫМИ

Оператор	Описание	Пример	Результат
+	Сложение	7 + 3	10
-	Вычитание	7 - 3	4
*	Умножение	7 * 3	21
/	Деление (истинное)	7 / 3	2.3333333333333335
**	Возведение в степень	7**3	343
//	Целочисленное деление	7 // 3	2
%	Остаток от деления	7 % 3	1

СТРОКИ

Строки в Python - упорядоченные последовательности символов, используемые для хранения и представления текстовой информации, поэтому с помощью строк можно работать со всем, что может быть представлено в текстовой форме.

Объявить строку можно при помощи кавычек или же функции **str** (если нужно перевести другой тип данных в строку)

Создание при помощи “”

```
s = "Hello world !"
print(s)
```

Создание при помощи **str**

```
s = str(737373)
print(s)
```

Создание при помощи ‘’

```
s = 'Hello world !'
print(s)
```

ПЕРЕМЕННЫЕ И ВЫВОД

Функция print

Исходный код

```
1 print(5)
2 print(2.5, 3)
3 print(1, 0.1, 5-3j)
```

Результат

```
5
2.5 3
1 0.1 (5-3j)
```

Переменные

Исходный код

```
1 a = 5
2 b1 = b2 = 3
3 c = (a + b1) * b2
4 print(c)
```

Результат

```
24
```

Комментарии

Исходный код

```
1 # Это комментарий
2 c = 299_792_458  # м/с
3 m = 1e-10        # кг
4 E = m * c**2     # Дж
5 print(E)
```

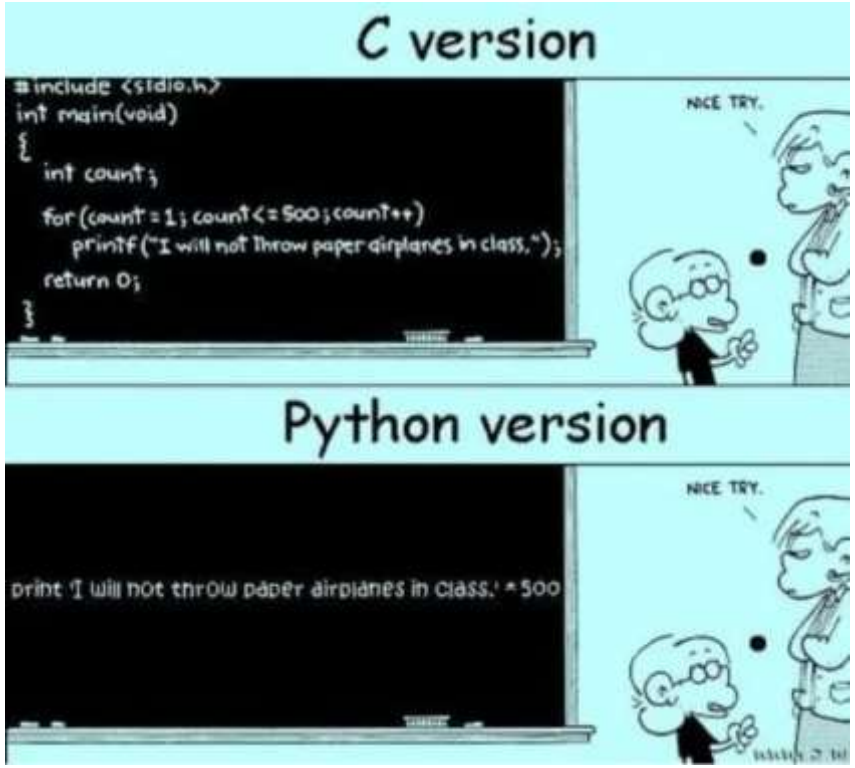
Результат

```
8987551.787368177
```

СТРОКИ (ОСОБЕННОСТИ)

Сравнение строк

```
1 a = "Some text"
2 b = 'Some text'
3 print(a, b)
4 print(a == b)
```



Кавычки внутри кавычек

```
1 a = "Can't use \"."
2 b = 'Can\'t use \'.'
3 print(a)
4 print(b)
```

Сложение строк

```
s1 = "Hello"
s2 = "World"
s3 = s1 + s2
print(s3)
```

Умножение строки на число

```
s1 = "A"
s2 = 25 * s1
print(s2)
```

СТРОКИ (МЕТОДЫ)

S.find(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или -1
S.rfind(str, [start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или -1
S.index(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или вызывает ValueError
S.rindex(str, [start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или вызывает ValueError
S.replace(шаблон, замена[, maxcount])	Замена шаблона на замену. maxcount ограничивает количество замен
S.split(символ)	Разбиение строки по разделителю
S.isdigit()	Состоит ли строка из цифр
S.isalpha()	Состоит ли строка из букв
S.isalnum()	Состоит ли строка из цифр или букв
S.islower()	Состоит ли строка из символов в нижнем регистре
S.isupper()	Состоит ли строка из символов в верхнем регистре
S.isspace()	Состоит ли строка из неотображаемых символов (пробел, символ перевода страницы ('\f'), "новая строка" ('\n'), "перевод каретки" ('\r'), "горизонтальная табуляция" ('\t') и "вертикальная табуляция" ('\v'))

СТРОКИ (МЕТОДЫ)

S.istitle()	Начинаются ли слова в строке с заглавной буквы
S.upper()	Преобразование строки к верхнему регистру
S.lower()	Преобразование строки к нижнему регистру
S.startswith(str)	Начинается ли строка S с шаблона str
S.endswith(str)	Заканчивается ли строка S шаблоном str
S.join(список)	Сборка строки из списка с разделителем S
ord(символ)	Символ в его код ASCII
chr(число)	Код ASCII в символ
S.capitalize()	Переводит первый символ строки в верхний регистр, а все остальные в нижний
S.center(width, [fill])	Возвращает отцентрованную строку, по краям которой стоит символ fill (пробел по умолчанию)
S.count(str, [start],[end])	Возвращает количество непересекающихся вхождений подстроки в диапазоне [начало, конец] (0 и длина строки по умолчанию)
S.expandtabs([tabsize])	Возвращает копию строки, в которой все символы табуляции заменяются одним или несколькими пробелами, в зависимости от текущего столбца. Если TabSize не указан, размер табуляции полагается равным 8 пробелам

СТРОКИ (МЕТОДЫ)

S.lstrip([chars])	Удаление пробельных символов в начале строки
S.rstrip([chars])	Удаление пробельных символов в конце строки
S.strip([chars])	Удаление пробельных символов в начале и в конце строки
S.partition(шаблон)	Возвращает кортеж, содержащий часть перед первым шаблоном, сам шаблон, и часть после шаблона. Если шаблон не найден, возвращается кортеж, содержащий саму строку, а затем две пустых строки
S.rpartition(sep)	Возвращает кортеж, содержащий часть перед последним шаблоном, сам шаблон, и часть после шаблона. Если шаблон не найден, возвращается кортеж, содержащий две пустых строки, а затем саму строку
S.swapcase()	Переводит символы нижнего регистра в верхний, а верхнего – в нижний
S.title()	Первую букву каждого слова переводит в верхний регистр, а все остальные в нижний
S.zfill(width)	Делает длину строки не меньшей width, по необходимости заполняя первые символы нулями
S.ljust(width, fillchar=" ")	Делает длину строки не меньшей width, по необходимости заполняя последние символы символом fillchar
S.rjust(width, fillchar=" ")	Делает длину строки не меньшей width, по необходимости заполняя первые символы символом fillchar
S.format(*args, **kwargs)	Форматирование строки

ЛОГИЧЕСКИЕ ОПЕРАЦИИ

Python	Математика	Значение
<	<	Меньше чем
<=	≤	Меньше или равно чем
==	=	Равно
>=	≥	Больше или равно чем
>	>	Больше чем
!=	≠	Не равно

Оператор	Описание
and	Логический оператор "И". Условие будет истинным, если оба операнда истина
or	Логический оператор "ИЛИ". Если хотя бы один из операндов истинный, то и все выражение будет истинным
not	Логический оператор "НЕ". Изменяет логическое значение операнда на противоположное

КОНСТРУКЦИИ IF, ELSE, ELIF

Простейший случай с **if**

```
if <условие>:
    <тело оператора>
```

Вариант с **else**

```
if <условие>:
    <тело оператора>
else:
    <тело оператора>
```

Вариант с **elif**

```
if <условие>:
    <тело оператора>
elif <условие>:
    <тело оператора>
```

Реальный пример

```
data = 0
if data > 0:
    print(data, "is a positive number.")
elif data == 0:
    print("The data is:", data)
else:
    print(data, "is a negative number.")
```

СПИСКИ

Списки в Python - упорядоченные изменяемые коллекции объектов произвольных типов (почти как массив, но типы могут отличаться).

Генерировать списки можно различными способами. Можно использовать встроенную функцию **list**, литерал [] или же генератора списков.

Для обращения к элементу списка, номер элемента передается в []. Важно помнить, что в **Python** нумерация начинается с 0!

Создание при помощи **list**

```
l = list('список')
# ['с', 'п', 'и', 'с', 'о', 'к']
```

Создание при помощи []

```
l = []
s = [1, 'h' , False]
```

Создание при помощи генератора

```
l = [c for c in 'список']
# ['с', 'п', 'и', 'с', 'о', 'к']
```

Обращение к элементу

```
s = [1, 'h' , False]
print(s[1])
```

СПИСКИ (МЕТОДЫ)

Метод	Что делает
list.append(x)	Добавляет элемент в конец списка
list.extend(L)	Расширяет список list, добавляя в конец все элементы списка L
list.insert(i, x)	Вставляет на i-ый элемент значение x
list.remove(x)	Удаляет первый элемент в списке, имеющий значение x. ValueError, если такого элемента не существует
list.pop([i])	Удаляет i-ый элемент и возвращает его. Если индекс не указан, удаляется последний элемент
list.index(x, [start [, end]])	Возвращает положение первого элемента со значением x (при этом поиск ведется от start до end)
list.count(x)	Возвращает количество элементов со значением x
list.sort([key=функция])	Сортирует список на основе функции
list.reverse()	Разворачивает список
list.copy()	Поверхностная копия списка
list.clear()	Очищает список

СПИСКИ (СРЕЗЫ)

Вывод второго элемента с конца

```
a = [1, 2, 3, 4, 5]
print(a[-2])
```

Вывод элементов с третьего по шестой

```
a = [1, 2, 3, 4, 5, 6, 7]
print(a[2:5])
```

Вывод элементов на четных позициях

```
a = [1, 2, 3, 4, 5]
print(a[::2])
```


```
a = [1, 2, 3, 4, 5]
print(a[::-1])
```

Вывод элементов на четных позициях со второго (не включая) по шестой в обратном порядке

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(a[7:1:-2])
```

СПИСКИ (РАСПАКОВКА)

Списки можно распаковывать, это позволяет помещать элементы списка в отдельные переменные. Стоит помнить, что при полной распаковке, количество переменных должно равняться количеству элементов в списке. Кроме того, распаковка позволяет объединять списки.

```
#Полная распаковка
b, c, d, e = a
print(b, c, d, e)

#Частичня распаковка
*b, c, d = a
print(b, c, d)
```

→

```
1 2 3 4
[1, 2] 3 4
```

Объединение списков

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
b = [4, 11, 17, 52]
print([*a, *b])
```

→

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 4, 11, 17, 52]
```

КОРТЕЖИ

Кортеж – по сути является неизменяемым списком. Для чего нужен? Защищает от случайных или намеренных изменений а так же экономит память. Плюс к этому, кортежи можно использовать как ключи словарей.

Объявить кортеж можно при помощи функции **tuple** или же ().

Кортежи поддерживают срезы и распаковку, но не поддерживают функции списков.

```
c = (5, 6, 7)
c_tuple = tuple("кортеж")

print(c, c_tuple)
```

Цикл while

Цикл while (“пока”) позволяет выполнить одну и ту же последовательность действий, пока проверяемое условие истинно.

Условие записывается до тела цикла и проверяется до выполнения тела цикла.

Как правило, цикл while используется, когда невозможно определить точное значение количества проходов исполнения цикла.

Синтаксис цикла **while**

```
while <условие>:
    <тело цикла>
```

Пример цикла **while**

```
a = 0
while a < 5:
    a = a + 1
    print(a)
```

Цикл for

Цикл for, также называемый циклом с параметром, в языке Питон богат возможностями.

В цикле for указывается переменная и множество значений, по которому будет пробегать переменная. Множество значений может быть задано списком, кортежем, строкой или диапазоном. В связке с **for** часто используют функцию **range**, создающую последовательность с определённым шагом.

Синтаксис цикла **for**

```
for <итератор> in <коллекция>:
    <тело цикла>
```

Пример цикла **for**

```
1 a = 0
2 for i in range(1, 5, 2):
3     a = a + i
4 print(a)  # Равно 4
```

Цикл while и for

Для программного выхода из цикла в языке **Python** используется слово **break.** Для того, чтобы пропустить итерацию цикла, необходимо использовать оператор **continue.**

Пропуск итерации

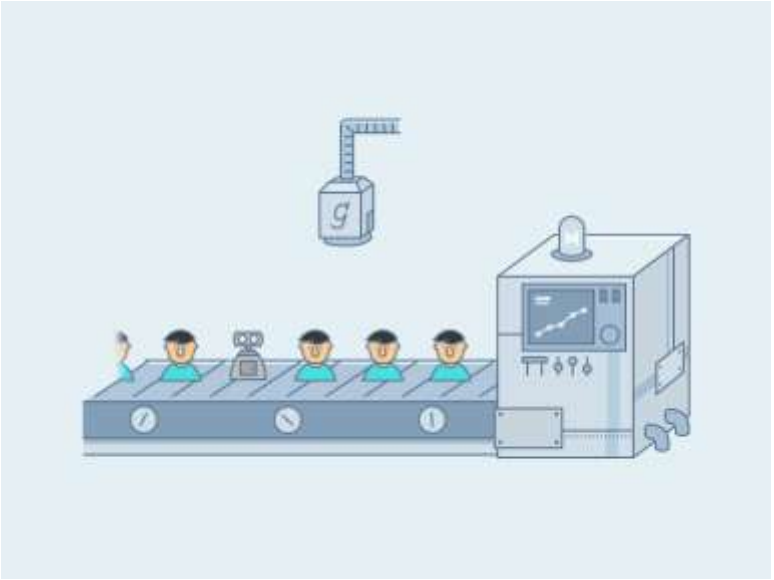
```
for i in range(1, 5):
    if i % 2:
        continue
    print(i)
```

Выход из цикла

```
i = 0
while True:
    if i == 5:
        break
    print(i)
    i += 1
```

break

continue



ГДЕ ТРЕНИРОВАТЬСЯ PYTHON?

SCHOOLSW3

https://www.schoolsw3.com/python/exercise.php

ПИТОНТЬЮТОР

https://pythontutor.ru/

CODE ACADEMY

https://www.codecademy.com/catalog

ЗАДАНИЕ ДЛЯ САМОКОНТРОЛЯ

- Задать список а равный [-3, -19, 4, -15, 2, -10, -3, 1, -11, 19]
- Вывести каждый третий элемент.
- Сделать срез списка от 7 элемента до начала списка.
- Развернуть список встроенной функцией.
- Отсортировать список.
- Создать программу на языке Python для решения квадратных уравнений, при этом при помощи управляющих конструкций if elif else, определять количество действительных корней уравнения.
- Создать программу, которая при помощи циклов, найдёт все простые числа в диапазоне от 1 до 200.
- Создать программу, которая будет подсчитывать процент гласных букв в строке, записанной на латинице.

СПАСИБО ЗА ВНИМАНИЕ!