




Основы программирования. Лекция 1

Python. Переменные. Типы данных. Память. Оформление кода

Что мы будем использовать в этом курсе?

- Linux
- Visual Studio Code
- git
- Python
- Для тестирования кода - pytest
- Для проверки оформления кода - линтер flake8 (возможно, ruff)

- Создан в 1991 г.
- Создатель: Гвидо Ван Россум
- Объектно-ориентированный
- Мультипарадигменный
- Интерпретируемый (cpython)
- Использует динамическую типизацию
- Официальный репозиторий: github.com/python
- Официальная документация: docs.python.org

Aug 2024	Aug 2023	Change	Programming Language		Ratings	Change
1	1			Python	18.04%	+4.71%
2	3	▲		C++	10.04%	-0.59%
3	2	▼		C	9.17%	-2.24%
4	4			Java	9.16%	-1.16%
5	5			C#	6.39%	-0.65%
6	6			JavaScript	3.91%	+0.62%

Zen of Python

Дзен языка python всегда можно посмотреть, вызвав интерпретатор и
выполнив код `import this`

1. Beautiful is better than ugly.
2. Explicit is better than implicit.
3. Simple is better than complex.
4. Complex is better than complicated.
5. Flat is better than nested.
6. Sparse is better than dense.
7. Readability counts.
8. Special cases aren't special enough to break the rules.
9. Although practicality beats purity.
10. Errors should never pass silently.
11. Unless explicitly silenced.
12. In the face of ambiguity, refuse the temptation to guess.
13. There should be one-- and preferably only one --obvious way to do it.
14. Although that way may not be obvious at first unless you're Dutch.
15. Now is better than never.

1. Красивое лучше, чем уродливое.
2. Явное лучше, чем неявное.
3. Простое лучше, чем сложное.
4. Сложное лучше, чем запутанное.
5. Плоское лучше, чем вложенное.
6. Разреженное лучше, чем плотное.
7. Читаемость имеет значение.
8. Особые случаи не настолько особые, чтобы нарушать правила.
9. При этом практичность важнее безупречности.
10. Ошибки никогда не должны замалчиваться.
11. Если они не замалчиваются явно.
12. Встретив двусмысленность, отбрось искушение угадать.
13. Должен существовать один и, желательно, только один очевидный способ сделать это.
14. Хотя он поначалу может быть и не очевиден, если вы не голландец.
15. Сейчас лучше, чем никогда.

Код на Python

```
1  # комментарии выделяются октоторпом (шарпом)
2  """
3      Комментарии бывают и многострочные,
4      как этот.
5  """
6  message = """
7      Но это может быть просто
8      длинная строка
9  """
10 my_number = 5 # переменные должны называться через зме
11 # а операторы - обособлены пробелами (почти всегда)
12 # название переменной должно быть коротким и должно од
13 # передавать назначение переменной
14 # длина строки кода - по разным стандартам не более 79
15 #
16 if my_number % 2 == 1:
```


Переменные

```
1 first_num = 5
2 second_num = 5.0
3 third_num = 5.
4 print('Первое число это', type(first_num))
5 print('Второе число это', type(second_num))
6 print('Третье число это', type(third_num))
7 print('Равны ли числа два и три? - ', second_num == third_num)
```

Результат выполнения:

```
1 Первое число это <class 'int'>
2 второе число это <class 'float'>
3 Третье число это <class 'float'>
4 Равны ли числа два и три? - True
```

Где хранятся наши данные?

Все объекты вашего программного кода хранятся в оперативной памяти, пока вы сами не решите записать их в энергонезависимую память.

Примерные скорости работы разных видов памяти:

- HDD - 200–300 МБ/с
- SSD - 600–700 МБ/с
- DRAM – 20-30 ГБ/с

Название типа	Байт	Диапазон значений	Название типа	Байт	Диапазон значений
short int	2	-32,768 to 32,767	signed char	1	-128 to 127
unsigned short int	2	0 to 65,535	unsigned char	1	0 to 255
unsigned int	4	0 to 4,294,967,295	float	4	1.2e-38 to 3.4e+38
int	4	-2,147,483,648 to 2,147,483,647	double	8	1.7e-308 to 1.7e+308
long int	4	-2,147,483,648 to 2,147,483,647	long double	16	0 to 1.1E+4932
unsigned long int	4	0 to 4,294,967,295			



Университет
Сириус
Колледж

Типы данных в языке Python

- Изменяемые
- Индексируемые
- Содержащие только уникальные элементы

Числа и строки - неизменяемые!
При этом строки - индексируемые.



Оператор Действие		Пример
+	Сложение двух операндов или унарный плюс	$x + y + 2$
-	Вычитание правого оператора из левого или унарный минус	$x - y - 2$
*	Умножение двух операндов	$x * y$
/	Деление левого операнда на правый (результат всегда типа float)	x / y
%	Остаток от деления левого операнда на правый	$x \% y$ (остаток от x / y)
//	Деление с округлением — деление, результат которого корректируется в меньшую сторону	$x // y$
**	Показатель степени — левый операнд возводится в значение правого операнда	$x ** y$



Оператор Действие		Пример
>	Больше чем: True, если левый операнд больше правого	$x > y$
<	Меньше чем: True, если левый операнд меньше правого	$x < y$
==	Равно: True, если операнды равны между собой	$x == y$
!=	Не равно: True, если операнды не равны между собой	$x != y$
>=	Больше или равно: True, если левый операнд больше или равен правому	$x >= y$
<=	Меньше или равно: True, если левый операнд меньше или равен правому	$x <= y$

Оператор	Действие	Пример
and	И: True, если оба операнда True	x and y
or	ИЛИ: True, если хотя бы один из операндов True	x or y
not	НЕ: True, если операнд False	not x

$$x = 10, y = 4$$

Оператор	Действие	Пример
&	Побитовое И	$x \& y = 0$ (00000000)
	Побитовое ИЛИ	$x y = 14$ (00001110)
~	Побитовое НЕ	$\sim x = -11$ (11110101)
^	Побитовое исключающее ИЛИ (XOR)	$x \wedge y = 14$ (00001110)
<<	Побитовый сдвиг влево	$x << 2 = 40$ (00101000)
>>	Побитовый сдвиг вправо	$x >> 2 = 2$ (00000010)

Оператор	Действие	Пример
<code>in</code>	True, если значение или переменная есть в последовательности	<code>5 in x</code>
<code>not in</code>	True, если значения или переменной нет в последовательности	<code>5 not in x</code>
<code>is</code>	True, если операнды идентичны (указывают на один объект)	<code>x is true</code>
<code>is not</code>	True, если операнды не идентичны (не указывают на один объект)	<code>x is not true</code>

- Неизменяемый итерируемый тип данных
- Могут быть объявлены разными способами
- Поддерживают синтаксис срезов (так же, как и списки)
- Поддерживают специальные символы
- Поддерживают интерполяцию
- Поддерживают форматирование
- Отдельные символы имеют кодировку по таблице ASCII

Таблица ASCII

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0.	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1.	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2.		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3.	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4.	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5.	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6.	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7.	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Строки. Способы объявления

```
1  # Строки могут быть заключены как в одинарные, так и в двойные кавычки
2  message = 'hello'
3  receipient = "world"
4
5  # Длинные строки можно заключить в любые кавычки, повторив кавычки в начале
6  greeting = """
7      - Hello
8          there!
9  """
10 answer = '''
11     - General
12         Kenobi!
13 '''
14
15 # Строки могут быть перенесены в коде с помощью символа \n
```



Индексы и срезы (slices)

- Получение символа из строки имеет следующий синтаксис:
`имя_переменной[индекс]`
- Срез имеет синтаксис: `имя_переменной[начало:конец:шаг]`

При этом начало включено, конец - нет!

Последовательность	a	b	c	d	e	f	g	Результат
Индексы	0(-7)	1(-6)	2(-5)	3(-4)	4(-3)	5(-2)	6(-1)	
[:]	+	+	+	+	+	+	+	abcdefg
[:: -1]	+	+	+	+	+	+	+	gfedcba
[:: 2]	+		+		+		+	aceg
[1 :: 2]		+		+		+		bdf
[: 1]	+							a
[-1 :]							+	g
[3 : 4]				+				d
[-3 :]					+	+	+	efg
[-3 : 1 : -1]			+	+	+			edc
[2 : 5]			+	+	+			cde

Символ	Описание
<code>\</code> в самом конце строки	Игнорируется, строка продолжается на новой строке
<code>\\</code>	Сам символ обратного слеша (остается один символ)
<code>\'</code>	Апостроф (остается один ')
<code>\"</code>	Кавычка (остается один символ ")
<code>\n</code>	Новая строка (перевод строки)
<code>\r</code>	Возврат каретки
<code>\t</code>	Горизонтальная табуляция

Если вы хотите, чтобы обратный слеш игнорировался, можете использовать **raw string** ("сырую" строку). Пример:

```
path = r'\path\to\folder\'
```




Интерполяция в строках (f-строки)

```
1  # интерполяция строк
2  message = 'hello there'
3  name = 'Obi-Wan Kenobi'
4  greeting = f'{name} said: {message}'
5  print (greeting)
6
7  # форматирование
8  greeting = '{0} said: {1}'. format (name, message)
9  print (greeting)
10
11 # более сложное форматирование
12 print (' {0:*^30.1f}'. format (1.18))
```

Результат:

```
1  Obi-Wan Kenobi said: hello there
2  Obi-Wan Kenobi said: hello there
3  *****1.2*****
```