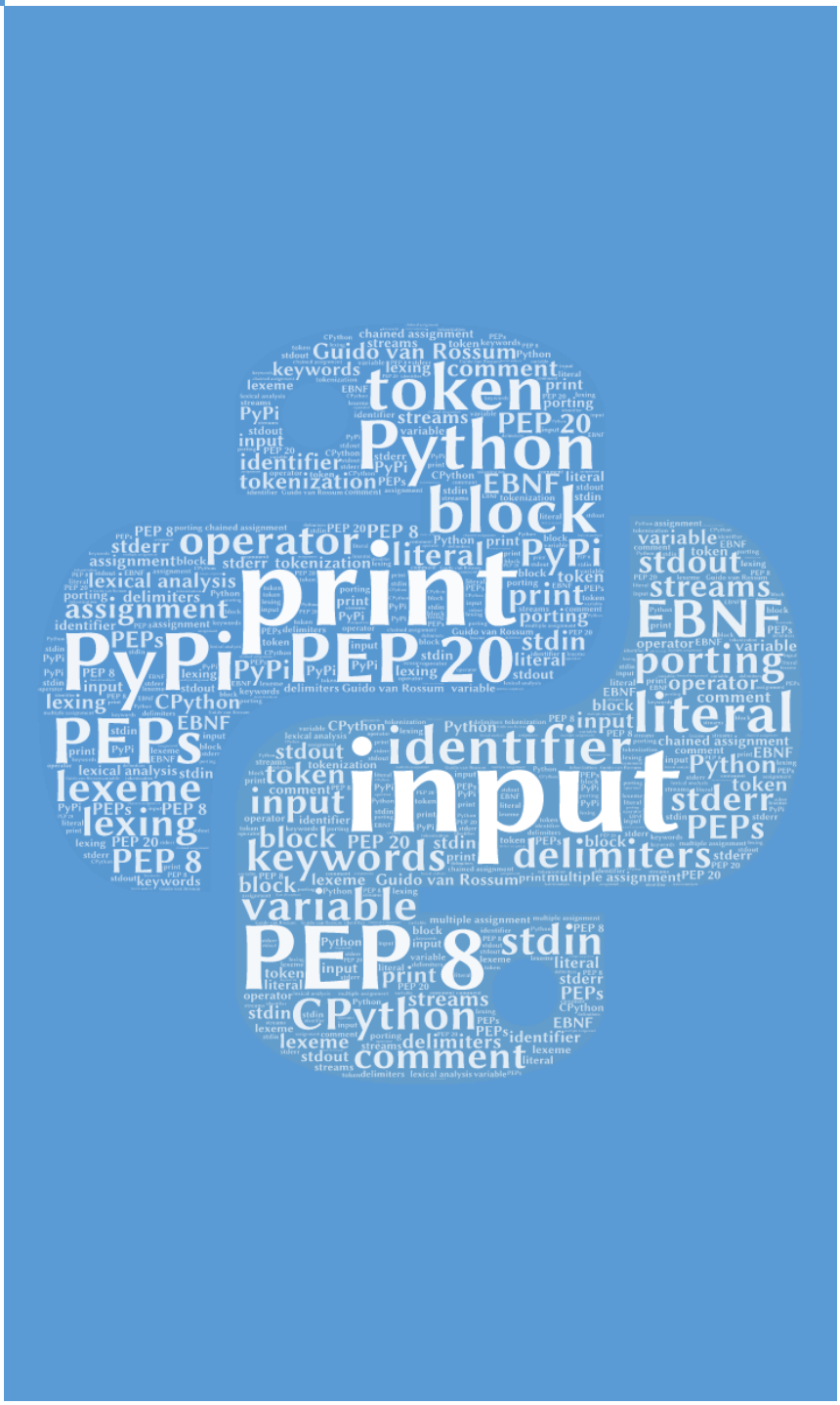
Университет «КРОК»
Кафедра компьютерных наук
Алгоритмизация и программирование. Лекции.
Автор: к.т.н., доцент Тимчук О.С.

4 ВВЕДЕНИЕ В ЯЗЫК ПРОГРАММИРОВАНИЯ PYTHON



4

Введение в язык программирования Python

4.1 Общая характеристика и особенности Python

Python – интерпретируемый высокоуровневый (3GL) язык программирования общего назначения. Концепция Python была разработана Guido van Rossum в конце 1980-х гг. Guido van Rossum, работая в голландском институте CWI (Stichting Mathematisch Centrum), разрабатывал Python как расширяемый скриптовый язык для распределенной операционной системы Amoeba. Свое название Python получил в честь популярного британского комедийного ТВ-шоу 1970-х гг. “Monty Python's Flying Circus” («Летающий цирк Монти Пайтона»). Первый релиз Python 0.9.0 был выпущен в феврале 1991 г. В 1995 году Guido van Rossum продолжил свою работу над Python в CNRI (Corporation for National Research Initiatives), где было выпущено несколько последующих версий языка. В мае 2000 года Guido и команда разработчиков ядра Python перешли в BeOpen.com для формирования команды BeOpen PythonLabs. В октябре 2002 г. был выпущен релиз Python 2.0. В октябре того же года команда PythonLabs переходит в Digital Creations (сейчас Zope Corporation). В 2001 году была создана некоммерческая организация Python Software Foundation (PSF) с целью содействия, защиты и расширения языка Python и его международного сообщества. В декабре 2008 г. был выпущен релиз Python 3.0. В настоящее время поддерживаются две линейки Python – Python v. 2.x Python v. 3.x.

Развитие Python происходит согласно чётко регламентированному процессу создания, обсуждения, отбора и реализации документов – предложения по развитию Python (Python Enhancement Proposals или PEPs, см. https://www.python.org/dev/peps/). Философия Python представлена в PEP 20 – The Zen of Python (см. https://www.python.org/dev/peps/pep-0020/) и состоит из 20 афоризмов:

- beautiful is better than ugly / красивое лучше, чем уродливое;
- explicit is better than implicit / явное лучше, чем неявное;
- simple is better than complex / простое лучше, чем сложное;
- complex is better than complicated / сложное лучше, чем запутанное;
- flat is better than nested / плоское лучше, чем вложенное;
- sparse is better than dense / разреженное лучше, чем плотное;
- readability counts / читаемость имеет значение;

- special cases aren't special enough to break the rules / особые случаи не настолько особые, чтобы нарушать правила;
- although practicality beats purity / при этом практичность важнее безупречности;
- errors should never pass silently / ошибки никогда не должны замалчиваться;
- unless explicitly silenced / если не замалчиваются явно;
- in the face of ambiguity, refuse the temptation to guess / встретив двусмысленность, отбрось искушение угадать;
- there should be one-- and preferably only one --obvious way to do it / должен существовать один — и, желательно, только один — очевидный способ сделать это;
- although that way may not be obvious at first unless you're dutch / хотя он поначалу может быть и не очевиден, если вы не голландец;
- now is better than never / сейчас лучше, чем никогда;
- although never is often better than *right* now / хотя никогда зачастую лучше, чем прямо сейчас;
- if the implementation is hard to explain, it's a bad idea / если реализацию сложно объяснить — идея плоха;
- if the implementation is easy to explain, it may be a good idea / если реализацию легко объяснить — идея, возможно, хороша;
- namespaces are one honking great idea -- let's do more of those! / пространства имён — отличная вещь! давайте будем делать их больше!

С начала выхода первой версии уже прошло более 25 лет и в настоящее время Python успешно используется во всем мире для разработки информационных систем в промышленности, в сфере услуг и науке. Рассмотрим основные пять преимуществ Python, которые делают его одним из самых популярных языков программирования:

1. Простота Python. Python является простым языком как для обучения, так и кодирования. Четкий и последовательный синтаксис Python и его модульность позволили размыть границы между пользователями и программистами – все больше ученых, инженеров, финансовых экспертов и других специалистов, имеющих небольшой опыт программирования, используют Python для решения своих конкретных повседневных технических задач. При этом большое количество различных конструкций языка позволяют разработчику сконцентрироваться не на синтаксисе программы, а на решении своей задачи - задача, которая требует в среднем двадцать строк кода на C и семь на Java, часто может быть выполнена с помощью только одной строки в Python.
2. Доступность Python. Python распространяется на условиях PSF лицензии (BSD-подобная пермиссивная лицензия на свободное ПО), которая совместима с GNU General Public License (GPL), что делает Python свободно используемым и распространяемым даже в коммерческих целях. Подробно с условиями лицензии можно познакомиться на официальном сайте https://docs.python.org/3/license.html.
3. Экосистема Python. Богатая и развитая экосистема Python предоставляет бесконечные возможности для разработки web-, desktop-, data science-, game-

приложений, организации доступа к базам данных, сетевого программирования и др. Все это возможно благодаря репозиторию программного обеспечения для Python (Python Package Index, PyPi, см. https://pypi.python.org/pypi). По данным счётчика на главной странице, репозиторий содержит данные о более чем 189 000 проектах.

4. Портируемость Python (англ. porting). Python позволяет выполнять адаптацию программы с сохранением интерфейса и функционала под другие платформы. Существуют порты на ряд стандартных платформ (например, Windows, Linux, Mac OS), специализированных платформ (например, Android, Windows Mobile) и устаревших платформ (например, MS-DOS) при условии отсутствия вызова системно-зависимых функций в исходном коде. По мере устаревания платформы её поддержка в основной ветви языка прекращается.
   Эталонной реализацией Python является интерпретатор CPython. Кроме стандартной реализации Python, существуют альтернативные решения, например, для виртуальной машины Java — Jython, для интеграции с платформой Microsoft .NET – IronPython.
5. Поддержка пользователей Python. Пользователям Python кроме свободно распространяемой документации высокого качества доступна также поддержка международного сообщества. На официальном сайте в разделе с документацией (см. https://www.python.org/doc/) доступны учебные материалы для пользователей разного уровня, FAQ, ссылки на учебники, обучающие аудио-и видео-ресурсы. В разделе сообщества (см. https://www.python.org/community/) доступна информация о каналах, группах и форумах, конференциях, посвященных Python, и многое другое.

Описание успешно реализованных Python-проектов в таких известных компаниях, как Mozilla Corporation, Google, SurveyMonkey, SMS Siemag AG доступно на официальном сайте Python в разделе Success Stories (см. https://www.python.org/success-stories/), а также на сайте https://brochure.getpython.info/.

4.2 Описание синтаксиса Python

Для формального описания правил построения исходного кода программы в языках программирования используют различные нотации. В настоящее время чаще всего используют нотацию расширенных форм Бэкуса-Наура (англ. Extended Backus–Naur Form, EBNF).

Нотация форм Бэкуса-Наура (англ. Backus–Naur Form, BNF) была впервые применена при описании языка Algol 60 (Peter Naur, 1960 г.). Развитие BNF привело к созданию нотации EBNF, которая впервые была применена при описании языков Modula-2 (Niklaus Emil Wirth, 1978 г.).

Описание синтаксиса языка с помощью EBNF состоит из набора правил, определяющих отношения между терминальными и нетерминальными символами. Терминальный

символ – отдельный символ или последовательность символов, являющихся с точки зрения синтаксиса неразрывным целым, не сводимым к другим символам (при описании выделяется одинарными или двойными кавычками). Нетерминальный символ – некоторая абстракция, по определенным правилам сводится к комбинации терминальных и/или других нетерминальных символов. Каждое правило состоит из нетерминального символа и EBNF -выражения, которые разделены знаком равенства. Выражение – соответствующая правилам EBNF комбинация терминальных и нетерминальных символов, операций. Правило завершается точкой. В таблице 4.1 приведено описание основных терминальных символов согласно международному стандарту ISO/IEC 14977.

Таблица 4.1 – Терминальные символы EBNF

Терминальный символ	Значение	Пример
,	бъединение	A, B — элемент A, за которым следует элемент B
=	определение	лексема «=» её описание
|	альтернатива	A | B — либо элемент A, либо B
(*…*)	комментарий	(*a simple program syntax *)
(…)	группировка	(A B) — группировка элементов
[…]	выбор	[A] — элемент A входит или не входит
{…}	повторения	{A} — ноль или более элементов A
-	исключение	A – B — A без элемента B
‘…’	терминальный символ	‘0’ — терминальный символ «цифра один»
“…”	терминальный символ	“1” — терминальный символ «цифра один»
*	повторения	A* — ноль или более элементов A
?...?	специальная последовательность	all_characters = ?all visible characters?
;	разделитель	A = B; C = D; — два выражения, разделенные разделителем

Рассмотрим пример описания такого элемента языка программирования, как «десятичное целое число» с помощью EBNF.

```
Sign = '+' | '-';

Digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';

Integer = [Sign] Digit {Digit};
```

В описании синтаксиса языка программирования есть начальные символы (start symbols), из которых выводятся все предложения языка. В синтаксисе языка Python определены следующие начальные символы: single_input, file_input, eval_input.

```
single_input= NEWLINE | simple_stmt | compound_stmt NEWLINE

file_input= (NEWLINE | stmt)* ENDMARKER

eval_input= testlist NEWLINE* ENDMARKER
```

Исходный код программы Python воспринимает как последовательность Unicode-символов (по умолчанию используется кодировка UTF-8). Конец логической строки определяется токеном NEWLINE, а файла – ENDMARKER.

Полное описание грамматики Python с помощью EBNF-выражений доступно в документации в разделе Full Grammar specification (см. https://docs.python.org/3/reference/grammar.html). В дальнейшем синтаксис языка Python будет описываться либо с помощью примеров, либо с помощью EBNF.

4.3 Алфавит и токены Python

Базовым элементом языка программирования является его алфавит, который состоит из множества неделимых символов. Алфавит Python состоит из следующего набора символов:

- буквы латинского алфавита ('a'...'z', 'A'...'Z');
- арабские цифры ('0'...'9');
- спецсимволы.

Последовательность символов из алфавита языка программирования, имеющая смысл для транслятора, называется лексемой (англ. lexeme). Лексемы в исходном тексте программы разделяются как минимум одним пробелом. В результате лексического анализа исходного текста программы (англ. lexical analysis, lexing or tokenization) для каждой лексемы строится токен (англ. token), которые затем передаются для синтаксического анализа программы.

В Python выделяют следующие типы токенов: идентификаторы, ключевые слова, литералы, операторы, разделители и специальные токены для обозначения конца логической строки, блока кода, комментариев. Рассмотрим подробнее типы токенов (см. https://docs.python.org/3/reference/lexical_analysis.html):

1. Идентификатор (англ. identifier) – последовательность символов (имя), позволяющая однозначно идентифицировать объект программы. Допустимые символы для идентификаторов: прописные и строчные латинские буквы, символ подчеркивания «_» и, за исключением первого символа, цифры от 0 до 9. В Python 3.x также введена поддержка дополнительных Unicode-символов (см. https://docs.python.org/3/reference/lexical_analysis.html#identifiers).

   Примечание:

   - рекомендуемая длина идентификатора – не более 79 символов;
   - не рекомендуется использовать символы «l» (строчная el), «O» (прописная oh) и «I» (прописная eye) для создания односимвольных идентификаторов, так как в некоторых шрифтах эти символы неотличимы от цифр 0 и 1, символа «|»;
   - при создании идентификаторов рекомендуется придерживать одного из следующих стилей: b (одиночная строчная буква), B (одиночная прописная

буква), lowercase (слово из строчных букв), lower_case_with_underscores (слова из строчных букв с подчеркиваниями), UPPERCASE (слово из прописных букв), UPPERCASE_WITH_UNDERSCORES (слова из прописных букв с подчеркиваниями), CapitalizedWords (слова с первыми прописными буквами).

Примеры идентификаторов:

```
>>> day = '12'          # 'day' – идентификатор.
>>> month = 'July'      # 'month' – идентификатор.
>>> year = '2019'# 'year' – идентификатор.
```

2. Ключевые слова (англ. keywords) – зарезервированные идентификаторы, назначение которых строго определено и использование в качестве обычных идентификаторов запрещено. Ниже представлены ключевые слова языка Python (см. https://docs.python.org/3/reference/lexical_analysis.html#keywords).

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

3. Литерал (англ. literal) – последовательность символов, с помощью которой представляют непосредственные значения разных типов данных. Рассмотрим примеры литералов некоторых встроенных типов данных Python.
   - пример литералов целого типа данных (int):

```
>>> 7
7
>>> 3
3
>>> 2147483647
2147483647
>>> 79228162514264337593543950336
79228162514264337593543950336
>>> 100_000_000_000
100000000000
>>> 0o177
127
>>> 0o377
255
>>> 0b100110111
311
>>> 0xdeadbeef
3735928559
>>> 0b_1110_0101
229
```

   - пример литералов вещественного типа данных (float):

```
>>> 3.14
3.14
>>> 10.
10.0
>>> .001
```

```
0.001
>>> 1e100
1e+100
>>> 3.14e-10
3.14e-10
>>> 0e0
0.0
>>> 3.14_15_93
3.141593
```

- пример литералов строкового типа данных (str):

```
>>> 'Python'
'Python'
>>> "Python"
'Python'
>>> 'Hello, '"world!"
'Hello, world!'
>>> 'Hello, \
'"world!"
'Hello, world!'
```

4. Оператор (operator) – спецсимвол или последовательность спецсимволов, которые обозначают некоторую операцию над данными. При этом смысл операции проявляется в зависимости от контекста, например, оператор «+» с данными целого типа будет указывать на операцию арифметического сложения, а с данными строкового типа – на операцию конкатенации (склеивания) строк. Ниже представлены операторы Python.

+	-	*	**	/	//	%	@
<<	>>	&	|	^	~		
<	>	<=	>=	==	!=		

5. Разделители (англ. delimiters) – спецсимволы, используемые для структурирования кода программы. Ниже представлены разделители Python.

(	)	[	]	{	}	
,	:	.	;	@	=	->
+=	-=	*=	/=	//=	%=	@=
&=	|=	^=	>>=	<<=	**=	

6. Конец логической строки. Python-программа состоит из нескольких логических строк, конец которых обозначается с помощью токена NEWLINE. Логическая строка может состоять из нескольких физических строк (например, при использовании составных операторов).
7. Блок кода (англ. block) - логически сгруппированный набор идущих подряд инструкций в исходном коде программы, например, тело условного оператора, цикла или функции. В Python для выделения блока кода используются отступы (пробелы или символы табуляции) вместо операторных скобок. Для формирования одного отступа рекомендуется использовать 4 пробела или один символ табуляции. Логические строки с одинаковым размером отступа формируют блок, и заканчивается блок в том случае, когда появляется логическая

строка с отступом меньшего размера. Блоки могут быть вложенными. Управление отступами осуществляется с помощью токенов INDENT и DEDENT.

8. Комментарий (comment) – языковая конструкция, используемая для пояснения исходного кода программы. Комментарий не влияет на ход выполнения программы. Комментарий либо начинается с символа ‘#’ и заканчивается символом завершения строки (однострочный комментарий), либо заключается между символами-скобками ''' и ''' или """ и """ (многострочный комментарий).
   Примечание:
   - рекомендуется комментарии оформлять в виде полноценных предложений: начало комментария начинать с прописной буквы (если только первым словом не является идентификатор, начинающийся со строчной буквы), а заканчивать точкой;
   - рекомендуется при написании комментариев использовать английский язык;
   - не рекомендуется писать комментарии, если они объясняют очевидное;
   - не рекомендуется писать комментарии в одной строке и кодом программы.

4.4 Оператор присваивания в Python

Фундаментальными понятиями в языке программирования являются «переменная» (англ. variable) и «операция присваивания» (англ. assignment).

Переменная представляет собой поименованную область памяти, в которой хранится объект. Другими словами, переменная представляет собой пару <имя переменной, значение>, где имя переменной является идентификатором, а значение – адресом объекта в памяти. Переменные используются для осуществления доступа к данным и изменения их значения в ходе выполнения программы. Создание переменной в Python невозможно без операции присваивания.

Операция присваивания — это механизм связывания, позволяющий динамически устанавливать или изменять связь между именем переменной и ее значением.

Формальная запись простой операции присваивания выглядит следующим образом

<выражение слева> <оператор присваивания> <выражение справа>,

где <выражение слева> – имя переменной,

<выражение справа> – значение, которое будет присвоено переменной.

<оператор присваивания> – оператор "=" (язык Python).

Примеры операции присваивания:

```
>>> i = 0
>>> x = [0, 1]
>>> s = 'Python'
```

В случаях, когда слева от оператора присваивания будут указаны не имена переменных, а, например, литерал или некоторое выражение, то будет сформирована синтаксическая ошибка. Примеры некорректного использования оператора присваивания:

```
>>> 'Python' = x
SyntaxError: can't assign to literal
>>> 6 + 7 = x
SyntaxError: can't assign to operator
```

Пример. Написать программу, которая меняет местами содержимое двух целочисленных переменных x и y.

```
>>> x = 123
>>> y = 321
>>> tmp = x
>>> x = y
>>> y = tmp
>>> x
321
>>> y
123
```

Рассмотрим состояния переменных на каждом этапе выполнения программы:

1	`>>> x = 123`	x = 123
2	`>>> y = 321`	x = 123, y = 321
3	`>>> tmp = x`	x = 123, y = 321, tmp = 123
4	`>>> x = y`	x = 321, y = 321, tmp = 123
5	`>>> y = tmp`	x = 321, y = 123, tmp = 123
6	`>>> x`	x = 321, y = 123, tmp = 123
7	`>>> y`	x = 321, y = 123, tmp = 123

Решение вида

```
>>> x = 123
>>> y = 321
>>> x = y
>>> y = x
```

будет ошибочным, так как содержимое переменной x будет потеряно после первого оператора присваивания. Для решения этой задачи необходимо воспользоваться промежуточной переменной, которая выполнит роль буфера для обмена (в данном случае переменная tmp).

Кроме простой операции присваивания, Python также поддерживает цепочечное и множественное присваивание.

1. Цепочечное присваивание (англ. chained assignment) – связывание одного значения с несколькими переменными.

```
>>> a = b = 10
>>> a
10
>>> b
10
```

2. Множественное присваивание (англ. multiple assignment) – одновременное связывание нескольких значений с несколькими переменными. При множественном присваивании количество переменных может быть меньше, чем количество значений. В таких случаях перед одной из переменных должен стоять символ «*» и тогда с этой переменной будут связаны все оставшиеся значения, сгруппированные в список.

```
>>> a, b = 10, 20
>>> a
10
>>> b
20

>>> a, *b = 5, 10, 20, 30
>>> a
5
>>> b
[10, 20, 30]
```

Учитывая особенности оператора присваивания в Python, рассмотрим различные варианты решения задачи обмена значений двух целочисленных переменных.

«Традиционное» решение»	`>>> tmp = x`
	`>>> x = y`
	`>>> y = tmp`
		решение выполняется за три операции с использованием дополнительной переменной (tmp)

«Экономное» решение»	`>>> x = x + y`
	`>>> y = x - y`
	`>>> x = x - y`
		решение выполняется за три операции без использования дополнительной переменной

Решение на Python	`>>> x, y = y, x`	обмен выполняется параллельно

4.5 Организация ввода / вывода данных в Python

При решении задач часто необходимо организовать ввод данных в программу из внешних источников, а также вывод результатов работы программы. В Python ввод / вывод данных производится потоками (англ. streams), т. е. последовательностями байтов. При операциях ввода байты направляются от источника (например, файла или некоторого устройства) в «промежуточный буфер» в оперативной памяти. В операциях вывода поток байтов направляется из «промежуточного буфера» на приемник (например, файл или некоторое устройство). Синхронизация «промежуточный буфер» с источником или приемником осуществляется операционной системой.

При запуске Python-программы автоматически открываются три «стандартных» потока:

- stdin – стандартный поток ввода, связанный по умолчанию с клавиатурой;
- stdout – стандартный поток вывода, связанный по умолчанию с экраном дисплея;

- stderr – стандартный поток сообщений об ошибках, связанный по умолчанию с экраном дисплея.

Остальные потоки создаются с помощью функций открытия файлов на период их чтения/записи/редактирования и уничтожаются с помощью функций закрытия файла.

Консольный потоковый ввод в Python осуществляется с помощью функции input( ).

input([prompt]) -> string,

где [prompt] – необязательный аргумент строкового типа – приглашение командной строки.

Функция считывает строку данных, полученную с потока ввода, и возвращает её без завершающего символа перевода каретки.

```
>>> s = input('--> ')
--> Monty Python's Flying Circus
>>> s
"Monty Python's Flying Circus"
```

Для преобразования строки, которую возвращает функция input(), к другому типу данных необходимо использовать функции приведения типов. Например, при использовании значения, полученного с помощью функции input( ), в арифметическом выражении без преобразования сгенерирует исключение TypeError.

```
>>> x = input('X=')
X=3
>>> y = input('Y=')
Y=4
>>> res = int(x) + int(y) + 10
>>> res
17
>>> res = x + y + 10
TypeError: can only concatenate str (not "int") to str
```

Консольный потоковый вывод в Python осуществляется с помощью функции print( ).

print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False),

где objects –выводимые в поток вывода объекты,

sep – разделитель между объектами (по умолчанию – пробел),

end – строка, выводимая после последнего объекта (по умолчанию - символ перевода строки \n),

file – поток вывода (по умолчанию – stdout),

flush – если параметр установлен в True, то поток будет сброшен в поток вывода принудительно..

Потоковый вывод в Python разрешён для всех объектов встроенных типов данных. Аргументами функции print могут быть литералы, имена переменных, выражения. Выводимые объекты выводятся с выравниванием по левой границе, начиная с позиции курсора.

```
>>> x, y = 10, 'Python'
>>> print(x, y)
10 Python
>>> print('x =', x, ';', 'y =', y)
x = 10 ; y = Python
>>> print(str(x) + y)
10Python
>>> print(x, y, sep='*')
10*Python
>>> print(x, y, sep='\n')
10
Python
```

4.6 Терминология

Портируемость

Расширенные формы Бэкуса-Наура

Алфавит

Лексема

Токен

Идентификатор

Ключевые слова

Литерал

Оператор

Разделитель

Блок кода

Комментарий

Переменная

Операция присваивания

Поток

Стандартный поток ввода

Стандартный поток вывода

Стандартный поток сообщений об ошибках

4.7 Контрольные вопросы и упражнения

Контрольные вопросы:

1. Опишите основные особенности языка Python.
2. Перечислите основные терминальные символы РБНФ, используемые для описания грамматики Python.
3. Сформулируйте понятие Python-программы, используя РБНФ выражения.
4. Дайте определение понятию «лексема». Перечислите основные типы токенов в Python.

5. Приведите примеры литералов встроенных типов данных Python.
6. Дайте определение понятию «переменная». Опишите основные свойства переменной в Python.
7. Дайте определение понятию «присваивание». Опишите основные особенности оператора присваивания в Python.
8. Приведите классификацию «стандартных» потоков ввода-вывода.
9. Опишите особенности организации консольного ввода данных в Python.
10. Опишите особенности организации консольного вывода данных в Python.

Вопросы для самостоятельной работы:

1. Основные возможности и особенности экосистемы Python.
2. Установка и управление программными Python-пакетами с помощью системы управления пакетами pip.
3. Правила оформления Python-кода согласно PEP-08. Проверка соответствия исходного кода программы правилам PEP8 в IDE PyCharm.
4. Инструменты управления отступами и блоками кода в PyCharm.
5. Механизмы перенаправления стандартных потоков ввода / вывода в Python.