Занятие 16. Комментарии. Типы данных. Функции, определяющие тип данных

План занятия

1. Комментарии

Комментариями в Python называются строки или части строк программы, начинающиеся с символа '#'. Такие строки не воспринимаются интерпретатором Python, как команды, а игнорируются. Комментарии используются для описания на естественном языке нетривиальных (неочевидных) частей программы.

Ещё один способ повысить читаемость программы — давать переменным имена, отражающие смысл (а иногда и тип) этих переменных. Не всегда разумно давать всем переменным длинные, «говорящие» названия. Но в любом случае стоит комбинировать оба эти метода (комментарии и имена переменных), чтобы программа оставалась понятной вне зависимости от времени, прошедшего с момента её написания.

Пример:

```
k = 2
n = int ( input () )
while n > 2:
        while (n % k) == 0: # пока можем делить на k
        print(k, end=' ␣ ')   # выписываем
n = n // k      # и делим
```

При составлении программ, использующих кириллицу, в начале программы рекомендуется размещать следующие комментарии:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
```

Если этого не сделать, то при использовании кириллических шрифтов, будет появляться сообщение об ошибке.

2. Типы данных

Типы данных, используемых в Python, также совпадают с другими языками – целые и вещественные типы данных; дополнительно поддерживается комплексный тип данных – с вещественной и мнимой частью (пример такого числа – 1.5J или 2j, где J представляет собой квадратный корень из -1). Python поддерживает строки, которые могут быть заключены в одинарные, двойные или тройные кавычки, при этом строки, как и в Java, являются immutable-объектами, т.е. не могут изменять свое значение после создания.

Есть в Python и логический тип данных bool с двумя вариантами значения – True и False. Однако в старых версиях Python такого типа данных не было, и, кроме того, любой тип данных мог быть приведен к логическому значению True или False. Все числа, отличные от нуля, и непустые строки или коллекции с данными трактовались как True, а пустые и нулевые значения рассматривались как False. Эта возможность сохранилась и в новых версиях Python, однако для повышения читаемости кода рекомендуется использовать для логических переменных тип bool. В то же время, если необходимо поддерживать обратную совместимость со старыми реализациями Python, то в качестве логических переменных стоит использовать 1 (True) или 0 (False).

Переменные в **Python** не нуждаются в обязательном их объявлении для выделения им памяти. Объявление (присваивание, назначение и т.п.) переменной и выделение памяти для хранения её значения выполняется в тот момент, когда им задаются какие-то данные (присваивание значения) этой переменной.

3. Стандартные типы данных

Данные, хранимые в памяти мог быть нескольких типов. Например, возраст человека хранится в численном виде, а его (её) адрес – в буквенно-

цифровом. У **Python**-а есть различные стандартные типы, по которым определяются допустимые операции над ними и методы хранения для каждого из них

В **Python** имеется пять таких стандартных типов:

- числа (numbers);
- строки (string);
- списки (list);
- кортежи (tuple);
- словари (dictionary).

4. Числа в Python

Числовые данные (*numeric data*) содержат цифровые значения.

Числовые объекты создаются при назначении им данные, например:

1	>>> var1 = 1
2	>>> var2 = 10

Можно удалить ссылку на такой объект с помощью оператора del:

1	del var1[,var2[,var3[....,varN]]]]

Удалять можно как одиночный объект (переменную), так и несколько сразу, например:

1	>>> var1 = 1
2	>>> print var1

3	1
4	>>> del var1

5	>>> print var1
6	Traceback (most recent call last):

7	File "<stdin>", line 1, in <module>
8	NameError: name 'var1' is not defined

1	>>> var1, var2, var3 = 1, 2, 3
2	>>> print var1, var2, var3

3	1 2 3
4	>>> del var1, var2, var3

5	>>> print var1, var2, var3
6	Traceback (most recent call last):

7	File "<stdin>", line 1, in <module>
8	NameError: name 'var1' is not defined

Python поддерживает четыре различных числовых типа данных:

Int – целые числа (integers);

long – большие целые числа, так же можно использоваться для восьмеричных (octal) или десятеричных (hexadecimal) чисел;

float – числа с плавающей точкой;

complex – комплексные числа.

Примеры

Ниже приведены несколько примеров чисел в различных типах:

int	long	float	complex
10	51924361L	0.0	3.14j

100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

В **Python** допустимо использовать букву l для обозначения типа long, но рекомендуется использовать только прописную L, что бы избежать возможной путаницы с цифрой 1.

5. Строки в Python

Строки (*strings*) в **Python** представляют из себя набор символов, заключённый в одинарные (' ') или двойные (" ") кавычки. Определённые группы из списка могут быть получены с помощью операторов `[ ]` или `[ : ]`, с указанием индекса, который всегда начинается с 0 и заканчивается индексом -1.

Знак (`+`) является оператором конкатенации (объединения), а символ (`*`) – оператором повторения. Например:

01	>>> str = 'Hello World!'
02	>>> print str

03	Hello World! # напечатать всю строку;
04	>>> print str[0] # напечатать только первый символ строки;

05	H

06	>>> print str[2:5]

07	llo # напечатать символы с 3-го по 5-ый;
08	>>> print str[2:]

09	llo World! # напечатать символы начиная с 3-го и до конца строки;
10	>>> print str * 2

11	Hello World!Hello World! # напечатать строку два раза;
12	>>> print str + "TEST"

13	Hello World!TEST # напечатать конкатенирующую (объединенную) строку.

6. Списки в Python

Списки (*lists*) являются наиболее универсальными из всех типов данных в **Python**. Список содержит элементы, разделённые запятыми и ограниченные квадратными скобками (**[ ]**). В некотором роде списки подобны массивам в **C**. Разница между ними заключается в том, что элементы, принадлежащие одному списку, могут относиться к разным типам данных.

Получить доступ к элементам списка можно с использованием операторов **[ ]** или **[:]** с указанием индекса, который начинается 0 и заканчивается -1. Как и в строках – знак (+) обозначает конкатенацию элементов, а символ (*) – его повторение. Например:

01	>>> list = ['abcd', 786 , 2.23, 'john', 70.2]

02	>>> tinylist = [123, 'john']

03	>>> print list
04	['abcd', 786, 2.23, 'john', 70.2] # напечатать полный список;

05	>>> print list[0] # вывести первый элемент списка;
06	abcd

07	>>> print list[1:3]
08	[786, 2.23] # вывести элементы, начиная со 2-го и заканчивая 3-им;

09	>>> print list[2:]
10	[2.23, 'john', 70.2] # вывести элементы списка начиная со 2-го и до конца;

11	>>> print tinylist * 2
12	[123, 'john', 123, 'john'] # вывести все элементы списка два раза;

13	>>> print list + tinylist
14	['abcd', 786, 2.23, 'john', 70.2, 123, 'john'] # вывести объединённые списки.

7. Кортежи в Python

Кортежи (*tuples*) - ещё один тип данных в **Python**, схожий со списками и содержит в себе элементы, разделённые запятыми. В отличии от списков – элементы в кортеже ограничиваются круглыми скобками **( )**.

Основное различие между списками (lists) и кортежами (tuples):

– Списки ограничиваются квадратными скобками (**[ ]**) и их элементы и размер могут изменены;

– Кортежи ограничиваются круглыми скобками (**( )**) и не могут быть изменены.

Кортежи можно представлять себе как списки, но в *read-only* “режиме”. Например:

```
>>> tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
>>> tinytuple = (123, 'john')
>>> print tuple
('abcd', 786, 2.23, 'john', 70.2) # вывести все элементы кортежа;
>>> print tuple[0]
abcd # вывести первый элемент кортежа;
>>> print tuple[1:3]
(786, 2.23) # вывести второй и третий элементы;
>>> print tuple[2:]
(2.23, 'john', 70.2) # выввести все элементы, начиная с третьего;
>>> print tinytuple * 2
(123, 'john', 123, 'john') # вывести все элементы два раза;
>>> print tuple[1:3] * 2
(786, 2.23, 786, 2.23) # вывести второй и третий элементы два раза;
>>> print tuple + tinytuple
('abcd', 786, 2.23, 'john', 70.2, 123, 'john') # вывести объединённые элементы кортежей;
```

Следующий пример не сработает с кортежем, так как мы попытаемся изменить данные в нём, но – сработает со списком:

```
t>>> tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
>>> list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
>>> tuple[2] = 1000
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment # сообщение об ошибке - кортеж не поддерживает переназначение элемента;
>>> list[2] = 1000
>>> print list[2]
```

8. Словари в Python

Словари (dictionaries) в Python являются своего рода хеш-таблицами. Они так же подобны ассоциативным массивам и хешам в Perl, и содержат пары ключ:значение. Ключ в словаре может содержать любой тип данных, используемый в Python, но, как правило, это цифры или строки. Значения же могут быть любыми случайными элементами.

Словари заключаются в фигурные скобки ({ }), а доступ к значениями элементов осуществляется с помощью квадратных скобок ([]). Например:

```
01          >>> dict = {}
02          >>> dict['one'] = "This is one"

03      >>> dict[2]   = "This is two"
04      >>> tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

05        >>> print dict['one']
06        This is one # вывести значение ключа one;

07        >>> print dict[2]
08        This is two # вывести значение ключа "2";

09    >>> print tinydict
10    {'dept': 'sales', 'code': 6734, 'name': 'john'} # отобразить полное
содержимое словаря;

11       >>> print tinydict.keys()
```

```
12          ['dept', 'code', 'name'] # вывести все ключи словаря;

13          >>> print tinydict.values()
14          ['sales', 6734, 'john'] # вывести все значения в словаре.
```

В словарях нет общего соглашения о последовательности элементов, но неверно говорить что “элементы не отсортированы” – они попросту не упорядочены (тавтология получается, в оригинале это звучит как “It is incorrect to say that the elements are “out of order”; they are simply unordered“).

9. Преобразование различных типов данных

Иногда может возникнуть необходимость в преобразовании данных между различными встроенными типами. Для этого – можно просто использовать тип данных как функцию.

В Python есть несколько встроенных функций для выполнения таких преобразований. Эти функции возвращают новый объект, представляющий собой преобразованное значение.

Function	Description
int(x [,base])	преобразовать x в целое; base указывает тип данных, если x – строка (см. примечание ниже);
long(x [,base])	преобразовать в long integer, base указывает тип данных, если x – строка;
float(x)	преобразовать x в число с плавающей точкой (floating-point number);
complex(real [,imag])	создать комплексное число;
str(x)	преобразовать объект x в строковый вид;
repr(x)	преобразовать объект x в выражение;

eval(str)	Evaluates a string and returns an object.
tuple(s)	преобразовать s в кортеж;
list(s)	преобразовать s в список;
set(s)	преобразовать s в set.
dict(d)	создать словарь; d должен представлять собой пару ключ:значение;
frozenset(s)	преобразовать s в frozen set.
chr(x)	преобразовать целое число в символ;
unichr(x)	преобразовать целое число в Unicode-символ;
ord(x)	преобразовать единичный символ в его числовое значение;
hex(x)	преобразовать целое число в шестнадцатеричное число;
oct(x)	преобразовать целое число в восьмеричное число.

Примечание к функции int(x [,base]):

допустимые типы base (основание системы счисления) – от 0 (десятеричная) до 16;

перевести строку с шестнадцатеричным числом в десятичное цифровое значение – тут base=0, т.е. десятеричное:

```
1        >>> print int("0xdeadbeef", 0)
2        3735928559
```

перевести строку с десятеричным числом в десятичное цифровое значение;

```
1        >>> print int("101", 0)
2        101
```

перевести строку с двоичным числом в десятеричное значение:

```
1    >>> print int("1100101",2)
2    101
```

перевести строку с шестнадцатеричным значением в десятичное:

```
1    >>> print int("65",16)
2    101
```