Тема 1. Синтаксис и структура программы. Типы данных и переменные. Ввод и вывод данных

1.1. Синтаксис и структура программы

Синтаксис Python отличается от других языков программирования. Главное отличие заключается в выделении блоков внутри кода. Чаще всего в других языках для этой цели используются либо фигурные скобки **{}**, либо конструкция **begin…end**. В Python для объединения строк в блок используются отступы (пробелы). Стандартом рекомендовано использовать **4 пробела**. В специальных редакторах для этой цели используется табуляция. Строки кода с одинаковым отступом относятся к одному блоку. Но если код не является вложенным, то отступ ставить нельзя, иначе возникнет ошибка.

Во многих языках программирования каждая инструкция должна заканчиваться точкой с запятой. В Python тоже можно ставить точки с запятой, но это не обязательно, так как по умолчанию каждая инструкция пишется с новой строки. Но если есть необходимость поставить несколько инструкций в одну строку, то точку с запятой нужно поставить после каждой инструкции, кроме последней (здесь она не обязательна).

Python «чувствителен» к регистру, поэтому инструкция, написанная в неправильном регистре, работать не будет.

В коде можно ставить комментарии – они не будут интерпретироваться, но будут служить подсказкой для разработчика. Для того чтобы закомментировать часть кода в одной строке, в его начале ставим знак **#**. Многострочного комментария по умолчанию в Python нет, но его можно создать, заключив текст в утроенные кавычки (""").

Пример:

```
x = 1; y = 2; z = x + y # это правильная запись
a = 2 b = 3 c = a + b # так писать нельзя
print(z) # правильная запись
Print(c) # так писать нельзя, с большой буквы Print неизвестен
         # интерпретатору
"""
многострочный
комментарий
"""
```

1.2.Типы данных

Стандартные типы данных

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

Основные **типы данных**, изучаемые в нашем курсе:

- **bool** – логический тип данных, может принимать значения True или False;
- **int** – целые числа, их размер ограничен размером оперативной памяти;
- **float** – вещественные числа, дробная часть отделяется точкой;
- **complex** – комплексные числа;
- **str** – Unicode-строки;
- **list** – списки;
- **tuple** – кортежи;
- **dict** – словари;
- **set** – множества.

1.3. Идентификаторы (переменные)

Идентификаторы в Python – это имена, используемые для определения («идентификации») переменных, функций, классов, модулей и других объектов. При инициализации в переменной сохраняется ссылка на объект (адрес объекта в памяти компьютера). Благодаря этой ссылке объект в дальнейшем можно изменять из программы.

В отличие от многих других языков программирования, в Python при создании переменной не указывают ее тип, он автоматически определяется присваиваемым значением. Также нет отдельного раздела для создания переменных – они создаются там, где могут понадобиться, причем им сразу же должно быть присвоено значение.

При создании имени переменной нужно соблюдать следующие **правила**.

- В качестве имени переменной нельзя использовать зарезервированные слова (например: if).
- В имени переменной используются только английские буквы (A-Z, a-z), цифры (0-9), знак подчеркивания (_).
- На первом месте может стоять только английская буква или знак подчеркивания.
- Имя переменной может состоять только из одной английской буквы.
- Имя переменной не может состоять только из одного знака подчеркивания, так как этот знак зарезервирован.
- Python «чувствителен» к регистру, то есть A1 и a1 это разные переменные.

Таблица зарезервированных слов

and	False	nonlocal
as	finally	not
assert	for	or
break	from	pass
class	global	raise

continue	if	return
def	import	True
del	in	try
elif	is	while
else	lambda	with
except	None	yield

При необходимости эту таблицу можно получить непосредственно из Python, используя следующий код:

```
import keyword # подключаем модуль
print(keyword.kwlist) # выводим на консоль
```

Обратите внимание, что в зарезервированных словах важен регистр написания. Поэтому, например, нельзя назвать переменную **if**, а вот **IF** уже можно, но все равно не рекомендуется использовать такие названия.

Аналогично нельзя в качестве имен использовать встроенные идентификаторы, такие как: **abs**, **pow** и т. д.

Получить список встроенных идентификаторов можно также из Python, используя следующий код:

```
import builtins # подключаем модуль
print(dir(builtins)) # выводим на консоль
```

Исходя из данных правил, приведем **примеры правильных и неправильных имен переменных**:

```
a = 1    # правильное создание переменной
A = 2    # правильное создание переменной,
         # причем a и A - это разные переменные
a1 = 3   # правильное создание переменной
a_1 = 4  # правильное создание переменной
1a = 5   # неправильное создание переменной
_a = 6   # правильное создание переменной, но так лучше не делать, этот
         # способ зарезервирован для других целей
or = 2   # неправильное создание переменной, зарезервированное имя
```

1.4. Присваивание значений переменным

Переменные в Python не нуждаются в обязательном объявлении для выделения им памяти. Объявление (присваивание, назначение и т. п.) переменной и выделение памяти для хранения ее значения выполняется в тот момент, когда вы задаете какие-либо данные (присваиваете значение) этой переменной. Для этого используется символ «**=**» **(**равно).

Рассмотрим несколько примеров:

a = 100 (здесь мы объявили целочисленную переменную **int**)

a = 100.0 (здесь мы объявили число с плавающей запятой **float**)

a = 'Python' (здесь мы объявили строковую переменную **str**)

a = True (здесь мы объявили логическую переменную **bool**)

1.5. Множественное присваивание

Выше мы рассмотрели присваивание значения отдельной переменной. Но в Python поддерживаются и так называемые множественные присваивания.

Рассмотрим несколько случаев:

a = b = c = 5 (здесь мы всем трем переменным присваиваем одно и тоже значение, в нашем случае это целое число 5)

a, b, c = 5, 4.5 ,'Python' (здесь мы всем переменным присвоили разные значения, а именно – переменной **a** присвоили целое число 5, переменной **b** присвоили дробное число 4.5, и переменной **c** присвоили строковое значение 'Python')

a, b, c = '123' В итоге переменная **a = '1'**, **b = '2'**, **c = '3'**. Но такой способ допустим, только если слева количество переменных совпадает с количеством элементов последовательности справа. Иначе получим ошибку.

a, b, *c = '12345' При несоответствии количества элементов слева и справа можно сохранить список из оставшихся элементов. Для этого перед именем переменной ставится *. В результате получим: **a = '1', b = '2', c = ['3', '4', '5'].** Здесь переменные **a** и **b** это строки, а переменная **c** – это список.

Последние два приема являются примерами распаковки. Более подробно приемы распаковки рассматриваются в соответствующем разделе.

Пример программы:

```
a=b=c=3
print(a)
print(b)
print(c)

a,b=1,2
print(a)
print(b)

a,b=b,a
print(a)
print(b)
```

Результат работы программы:

```
3
3
3
1
2
2
1
```

1.6. Проверка типа данных

Python в любой момент времени при присваивании нового значения изменяет тип переменной в соответствии с присваиваемым значением.

Пример:

```
a = 1   # число int
a = '1' # строка str
```

Определить на какой тип ссылается переменная, позволяет функция **type(<**имя переменной>).

```
a = 5
print(type(a))
```

```
<class int>
```

1.7. Преобразование различных типов данных

В Python есть несколько встроенных функций для преобразования типов данных. Мы рассмотрим только необходимые нам для обучения функции.

int(s[,base])

Преобразует **s** в целое число. Необязательный параметр **base**, приведенный в квадратных скобках, указывает на **основание системы счисления** (от 2 до 36). Результат работы зависит от типа вводимых данных.

Для понимания принципа работы рассмотрим несколько **примеров**:

1. Передаем в качестве параметра дробное число

   int(1.5)

   В данном случае функция просто отсекает дробную часть и получает целое число. В результате получим **1**.

2. Передаем то же число, но уже в виде строки

   int('1.5')

   В данном случае получим ошибку: **invalid literal for int() with base 10: '1.5'**

3. Передаем двоичное число без указания основания

   int('101')

   В результате получим десятичное число **101**.

4. Передаем двоичное число с указанием основания

   int('101', 2)

   В результате получим десятичное число **5**.

   Аналогично и с другими системами счисления.

long(s[,base])

Преобразует **s** в длинное целое число. Необязательный параметр **base**, указанный в квадратных скобках, указывает на основание системы счисления (от 2 до 36). Результат работы зависит от типа вводимых данных.

Работает аналогично функции **int**.

float(s)

Преобразует **s** в число с плавающей точкой.

Пример:

float(2)

В результате получим дробное число **2.0**.

float('2.5')

В результате получим дробное число **2.5**.

str(s)

Преобразует **s** в строковое значение.

Пример:

str(5)

В результате получим строку из одного символа **'5'**.

ord(s)

Преобразует символ в его код.

Пример: в качестве параметра передаем большую латинскую **A**.

ord('A')

В результате получим число **65**.

chr(s)

Преобразует целое число (код символа) в символ.

Пример:

chr(65)

В результате получим большую латинскую букву **A**.

bin(s)

Преобразует целое десятичное число в двоичное.

Пример:

bin(5)

В результате получим двоичное число **101** в виде **0b101**, где **0b** указывает на то, что число двоичное.

oct(s)

Преобразует целое десятичное число в восьмеричное.

Пример:

oct(15)

В результате получим восьмеричное число **17** в виде **0o17**, где **0o** указывает на то, что число восьмеричное.

hex(s)

Преобразует целое десятичное число в шестнадцатеричное.

Пример:

hex(15)

В результате получим шестнадцатеричное число **f** в виде **0xf**, где **0x** указывает на то, что это шестнадцатеричное число.

1.8. Ввод и вывод данных

Ввод данных

Мы рассмотрели, каким образом можно присваивать значения переменным непосредственно в коде программы. Но это не всегда удобно и целесообразно. Чаще приходится вводить пользовательские данные с клавиатуры после запуска программы.

Для ввода данных используется специальная функция **input()**, которая считывает введенную с клавиатуры строку и возвращает ее значение.

Пример:

```
a = input() # считываем данные с клавиатуры
```

В данном случае введенная с клавиатуры строка присвоится нашей переменной **a**. Нужно понимать, что этот способ возвращает именно строковые данные, т. е. если мы таким образом введем с клавиатуры, например, «**2**», то получим не число **2**, а символ **'2'**. Поэтому после ввода данных, если нам нужны не символьные данные, их нужно конвертировать в соответствующий тип.

Пример 1:

```
a = input()
print(a + a)
```

В данном случае, если мы введем с клавиатуры число **2**, оно будет рассматриваться интерпретатором как символ (строка) **'2'**, и в результате выполнения программы мы получим строку **'22'.**

Пример 2:

```
a = input() # считываем данные с клавиатуры
a = int(a)  # конвертируем тип данных
print(a + a)
```

В данном случае, если мы введем с клавиатуры число **2**, оно будет рассматриваться интерпретатором как целое число **2**, и в результате выполнения программы мы получим число **4**.

Обратите внимание на скобки в функции **input()**, если их оставить пустыми, то на экране будет просто моргать курсор, и программа будет ожидать ввода. А если в скобках указать текст, то он выведется на экран.

Пример:

```
a = input('Введите число: ')
```

Мы увидим на экране фразу «Введите число:», а после нее будет мигать курсор ввода.

Вывод данных

Для вывода данных используется функция **print()**. В скобочках указывается переменная, значение которой мы выводим.

Пример:

```
a = 1
print(a)
```

```
1
```

Функция **print()** может выводить не только значения переменных, но и значения любых выражений.

Пример:

```
a = 3
print(a + 2)
```

```
5
```

Также при помощи функции **print()** можно выводить значение не одного, а нескольких выражений, для этого нужно перечислить их через запятую.

Пример:

```
a = 1
b = 2
print(a, '+', b, '=', a + b)
```

```
1 + 2 = 3
```

Сначала выводится значение переменной **a**, затем знак **'+'**, затем значение переменной **b**, затем знак **'='**, наконец, значение суммы **a + b**.

Обратите внимание, выводимые значения разделяются одним пробелом. Но такое поведение можно изменить: или разделять выводимые значения двумя пробелами, любым другим символом, любой другой строкой, выводить их в отдельных строках или никак не разделять. Для этого нужно функции **print()** передать специальный именованный параметр, называемый **sep**, равный строке, используемый

в качестве разделителя (**sep** — аббревиатура от слова **separator**, т. е. разделитель). **По умолчанию** параметр **sep** равен строке из одного пробела и **между значениями выводится пробел**. Чтобы использовать в качестве разделителя, например, символ двоеточия, нужно передать параметр **sep**, равный строке **':'**.

Пример:

```
a = 2
b = 3
c = 4
print(a, b, c, sep=':') # вывод с указанием разделителя
```

```
'2:3:4'
```

А для того, чтобы совсем убрать разделитель при выводе, нужно передать параметр **sep**, равный пустой строке:

Пример:

```
print(a, '+', b, '=', a + b, sep='')
```

Для того, чтобы значения выводились **с новой строки**, нужно в качестве параметра **sep** передать строку, состоящую из специального символа новой строки, которая задается так:

```
print(a, b, sep='\n')
```

Символ обратного слеша в текстовых строках является указанием на обозначение специального символа, в зависимости от того, какой символ записан после него. Наиболее часто употребляется символ новой строки **'\n'**. А для того, чтобы вставить в строку сам символ обратного слеша, нужно повторить его два раза: **'\\'**.

Вторым полезным именованным параметром функции **print** является параметр **end**, который указывает на то, что выводится после вывода всех значений, перечисленных в функции **print()**. По умолчанию параметр **end** равен **'\n'**, то есть следующий вывод будет происходить с новой строки. Этот параметр также можно исправить. Например, для того, чтобы **убрать все дополнительные выводимые символы,** можно вызывать функцию **print()** так:

```
print(a, b, c, sep='', end='') # убираем разделитель и перенос строки
```

Если нам нужно разными функциями **print** вывести в одну строку, то используем параметр **end**

```
print('a', end='')
print('b', end='')
```

```
ab
```

Для вывода результатов программы вместо функции **print()** можно использовать метод **write()** объекта **sys.stdout**. Но сначала нужно подключить модуль **sys**.

```
import sys             # подключаем модуль
sys.stdout.write('a') # выводим на экран
```

Более подробное изучение этого метода выходит за рамки данного курса.

1.9. Форматированный вывод

Выше мы рассмотрели простейший способ вывода строк. Но данный способ не всегда удобно использовать, более того, есть ситуации, когда при помощи него невозможно вывести данные в нужном виде. В Python для форматирования строки можно воспользоваться оператором **%,** а также методом **format()**. Более предпочтительным считается метод **format()**, но все же разберем оба способа.

Форматирование при помощи оператора %

Данный способ позволяет в строке слева от оператора **%** указать места подстановки объектов с указанием формата вывода. Затем на эти места подставляются соответствующие объекты, указанные справа от оператора **%**. Если в строке используется одна подстановка, и она является переменной, то ее можно указывать без скобок. Если подстановок несколько, то они заключаются в круглые скобки и разделяются запятыми. Если же в качестве подстановки используется выражение, его тоже нужно заключать в круглые скобки.

Основные форматы вывода для оператора %

Формат	Описание
'%d'	Десятичное целое число.
'%o'	Число в восьмеричной системе счисления.
'%x'	Число в шестнадцатеричной системе счисления с буквами в нижнем регистре.
'%X'	Число в шестнадцатеричной системе счисления с буквами в верхнем регистре.
'%f', '%F'	Число с плавающей точкой в обычном формате.
'%e'	Число с плавающей точкой с экспонентой в нижнем регистре.
'%E'	Число с плавающей точкой с экспонентой в верхнем регистре.
'%c'	Символ либо код символа.
'%s'	Строка.

Пример:

```
print('1/3=%d' %(1/3))
print('1/3=%f' %(1/3))
print('1/3=%e' %(1/3))
print('1/3=%s' %(1/3))
print('Число 9 в восьмеричной системе = %o' %(9))
print('Число 10 в шестнадцатеричной системе = %X' %(10))
print('%c' %'A')
print('%c' %65)
```

Результат:

```
1/3=0
1/3=0.333333
1/3=3.333333e-01
1/3=0.3333333333333333
Число 9 в восьмеричной системе = 11
Число 10 в шестнадцатеричной системе = A
A
A
```

При форматированном выводе после символа **%** в указании формата можно указывать дополнительные настройки. В данном разделе мы рассмотрим только некоторые из них, остальные предлагаем разобрать самостоятельно.

Параметр	Описание
Флаг	Указывает на то, чем будут заполняться «лишние» свободные позиции при выводе. Принимает следующие значения: '0' – свободное место заполняется нулями; " – свободное место заполняется пробелами слева; '-' – свободное место заполняется пробелами справа.
Минимальное количество знаков на вывод	Десятичное число, указывающее количество позиций на вывод. Если выводимое значение содержит меньше символов, остальные заполняются по определенному правилу, которое указывается в параметре «**флаг».**
Точность вывода	Желаемая точность вывода. При указании точности сначала ставится точка, а затем уже число, указывающее на саму точность (для чисел – количество знаков после запятой, для строк – количество выводимых символов).

Пример:

```
print("'%.2f'" %(1/3))
print("'%10.3f'" %(1/3))
print("'%-10.2f'" %(1/3))
print("'%010d'" %(5))
print("'%10s'" %('Python'))
print("'%.2s'" %('Python'))
```

Результат:

```
'0.33'
'     0.333'
'0.33      '
'0000000005'
'    Python'
'Py'
```

Форматирование при помощи метода format()

Метод форматированного вывода **format()** позволяет устанавливать в определенные позиции строки значения, указанные в качестве его аргументов. Места подстановки отмечаются фигурными скобками «{}». Скобки могут быть как пустыми, так и с параметрами. Если скобки пустые, то аргументы подставляются в скобки в порядке следования, т. е. в первую скобку в строке подставляется первый аргумент, во

вторую – второй и т. д. Если в скобки проставить числа (номера), то на их места будут подставляться аргументы под указанным номером. Нумерация аргументов начинается с 0. Также могут использоваться и именованные аргументы. Для лучшего понимания рассмотрим соответствующий пример.

Пример:

```
print('Привет, {}!'.format('Катя'))
s = 'Привет, {}!'
n = 'Катя'
print(s.format(n))
print('{} {} {}'.format('Мама', 'Мыла', 'Раму'))
print('{1} {0} {2}'.format('Мама', 'Мыла', 'Раму'))
print('{a} + {b} = {c}'.format(a=1, b=2, c=3))
```

Результат:

```
Привет, Катя!
Привет, Катя!
Мама Мыла Раму
Мыла Мама Раму
1 + 2 = 3
```

При построении строки в фигурных скобках также можно указывать, в каком формате должно выводиться подставляемое значение. Для этого нужно в скобках поставить знак «:» (двоеточие), а после него указать соответствующий формат. Здесь будем использовать те же форматы, что и при работе с оператором **%**. Обратите внимание на то, что в этом случае перед форматом не ставится знак «%».

Основные форматы вывода для метода fromat()

Формат	Описание
d	Десятичное целое число.
o	Число в восьмеричной системе счисления.
x	Число в шестнадцатеричной системе счисления с буквами в нижнем регистре.
X	Число в шестнадцатеричной системе счисления с буквами в верхнем регистре.
f, F	Число с плавающей точкой в обычном формате.
e	Число с плавающей точкой с экспонентой в нижнем регистре.
E	Число с плавающей точкой с экспонентой в верхнем регистре.
c	Код символа.
s	Строка.

Пример:

```
print('1/3 = {0:f}'.format(1/3))
print('1/3 = {0:e}'.format(1/3))
print('Число 9 в восьмеричной системе = {0:o}'.format(9))
print('Число 10 в шестнадцатеричной системе = {0:X}'.format(10))
print('{0:c}'.format(65))
```

Результат:

```
1/3 = 0.333333
1/3 = 3.333333e-01
Число 9 в восьмеричной системе = 11
```

```
Число 10 в шестнадцатеричной системе = A
A
```

При использовании метода **format()** нужно быть внимательным с типом данных. К примеру, если в операторе **%** к дробному числу применить флаг форматирования **d**, то выведется целая часть числа, а при использовании метода **format()** будет выдана ошибка.

Пример:

```
print('1/3 = {0:d}'.format(1/3))
```

```
builtins.ValueError: Unknown format code 'd' for object of type 'float'
```

Та же ситуация при попытке применить флаг форматирования **s** к числовому значению.

Пример:

```
print('1/3 = {0:s}'.format(1/3))
```

```
builtins.ValueError: Unknown format code 's' for object of type 'float'
```

При использовании метода **format()** также можно указывать точность вывода. Для этого после знака «:» (двоеточие) в фигурных скобках нужно поставить:

1) число;
2) точку и число;
3) число, точку и число.

В первом случае указывается минимальное количество позиций на вывод, во втором случае указывается точность, а в третьем случае указывается минимальное количество позиций на вывод и точность.

Пример:

```
print("'1/3 = {0:10f}'".format(1/3))
print("'1/3 = {0:.2f}'".format(1/3))
print("'{0:.2s}'".format('Python'))
print("'1/3 = {0:10.2f}'".format(1/3))
```

Результат:

```
'1/3 =   0.333333'
'1/3 = 0.33'
'Py'
'1/3 =       0.33'
```

Дополнительные опции метода format()

Метод формат позволяет также управлять выравниванием текста при выводе, указывать заполнитель для пустых позиций, а также управлять отображением знака «+» для положительных чисел.

Дополнительные флаги для метода fromat()

Формат	Описание
'<'	Выравнивание по левому краю.
'>'	Выравнивание по правому краю.
'='	Заполнители ставятся после знака. Работает только для чисел.
'^'	Выравнивание по центру.
'+'	Знак добавляется как для отрицательных так и для положительных чисел.
'-'	Для отрицательных чисел ставится знак «-», для положительных нет.
' '	Для отрицательных чисел ставится знак «-», для положительных пробел.

Пример:

```
print("'1/3 = {0:<15f}'".format(1/3))
print("'1/3 = {0:>15f}'".format(1/3))
print("'1/3 = {0:^15f}'".format(1/3))
print("'1/3 = {0:*^15f}'".format(1/3))
print("'1/3 = {0:*=15f}'".format(-1/3))
print("'{0:+5}:{1:+5}'".format(3,-3))
print("'{0:5}:{1:5}'".format(3,-3))
print("'{0: }:{1: }'".format(3,-3))
```

Результат:

```
'1/3 = 0.333333       '
'1/3 =        0.333333'
'1/3 =    0.333333    '
'1/3 = ***0.333333****'
'1/3 = -******0.333333'
'   +3:   -3'
'    3:   -3'
' 3:-3'
```