ЗАНЯТИЕ 7

Переменные, типы данных и операторы

Сегодня на занятии

Узнаем

- правила работы с переменным и типами данных
- правила преобразования типов данных
- базовые операторы

Переменные в Python

Переменные предназначены для хранения данных.
Именно с помощью переменных компьютер запоминает информацию, с которой в дальнейшем работает программа.

Технически, переменная является ссылкой на ячейку памяти, в которой храниться значение.

Переменная состоит из двух частей:

- Имя переменной
- Значение переменной

Имя – это идентификатор переменной. В программе не может быть двух переменных с одинаковыми именами, но при этом разными значениями.

Переменные в Python

ВАЖНО!

В имени переменной нельзя использовать пробелы, точки или запятые. Чтобы дать имя из двух слов, используется нижние подчеркивание «_» или каждое новое слово начинается с заглавной буквы:

myName

my_name

Имя переменной в Python

Название переменной в Python:

- должно начинаться с алфавитного символа или со знака подчеркивания и
- может содержать алфавитно-цифровые символы и
- может содержать знак подчеркивания.

- Название переменной не должно совпадать с названием ключевых слов языка Python:

```
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
```

Типы наименования переменных

В Python применяется два типа наименования переменных: **camel case** и **underscore notation**.

Camel case подразумевает, что каждое новое подслово в наименовании переменной начинается с большой буквы. Например:

```
1 userName = "Tom"
```

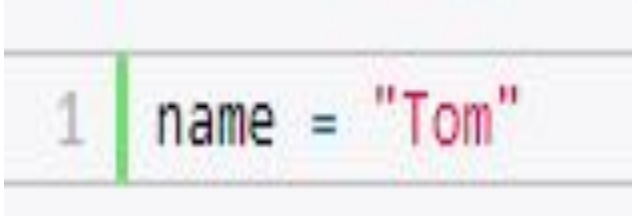
Underscore notation подразумевает, что подслова в наименовании

переменной разделяются знаком подчеркивания. Например:

```
1 user_name = "Tom"
```

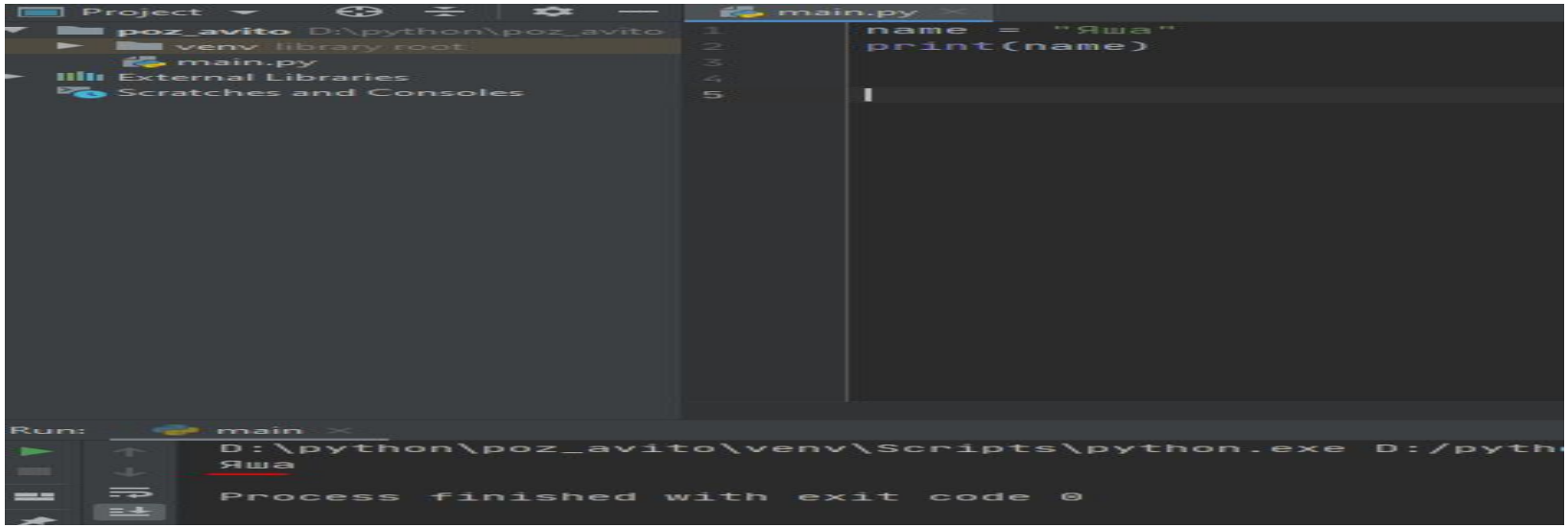
Определение переменной

Создание переменной в коде называется определением.
Для определения переменной необязательно объявлять к какому типу данных она относится.



Пример вывода значения переменной

Определив переменную, мы можем использовать её в программе. Например, попытаться вывести ее содержимое на консоль с помощью встроенной функции print:



Значение переменной

Отличительной особенностью переменной является то, что мы можем менять ее значение в течение работы программы:

```
1  name = "Tom"   # переменной name равна "Tom"
2  print(name)    # выводит: Tom
3  name = "Bob"   # меняем значение на "Bob"
4  print(name)    # выводит: Bob
```

Типы данных

Тип данных – описание того, какого рода данные хранятся в переменной.

Представим ситуацию, что в двух переменных хранятся разные типы данных: строка и число. В программе встречается инструкция – сложение двух этих переменных. Когда интерпретатор дойдёт до этой строчки, программа аварийно завершится, поскольку не только компьютер не знает как складывать строки и числа, но и человек.

Примеры типов данных

Переменная хранит данные одного из типов данных:

- **int** (integer) – число
- **float** (плавающая точка) – дробное число
- **str** (string) – строка
- **bool** (булева функция) – True или False (правда или ложь / из двоичной логики)

Это конечно же не все типы данных, которые существуют в языке Python, однако они являются базовыми.

Целочисленный тип данных

Тип **int** представляет целое число, например, 1, 4, 8, 50.
Пример:

```
1  age = 21
2  print("Возраст:", age)    # Возраст: 21
3
4  count = 15
5  print("Количество:", count) # Количество: 15
```

По умолчанию стандартные числа расцениваются как числа в **десятичной системе**. Но Python также поддерживает числа в двоичной, восьмеричной и шестнадцатеричной системах.

Другие системы счисления

Для указания, что число представляет **двоичную систему**, перед числом ставится префикс **0b**:

```
a = 0b11
b = 0b1011
c = 0b100001
print(a)    # 3 в десятичной системе
print(b)    # 11 в десятичной системе
print(c)    # 33 в десятичной системе
```

Для указания, что число представляет **восьмеричную систему**, перед числом ставится префикс **0o**:

```
a = 0o7
b = 0o11
c = 0o17
print(a)    # 7 в десятичной системе
print(b)    # 9 в десятичной системе
print(c)    # 15 в десятичной системе
```

Другие системы счисления

Для указания, что число представляет **шестнадцатеричную систему**, перед числом ставится префикс **0x**:

```
a = 0x0A
b = 0xFF
c = 0xA1
print(a)    # 10 в десятичной системе
print(b)    # 255 в десятичной системе
print(c)    # 161 в десятичной системе
```

Стоит отметить, что в какой бы системе мы не передали число в функцию **print** для вывода на консоль, оно по умолчанию будет выводиться **в десятичной системе**.

Вещественный тип данных

Тип float представляет число с плавающей точкой, например, 1.2 или 34.76. В качесте разделителя целой и дробной частей используется точка.

```
1  height = 1.68
2  pi = 3.14
3  weight = 68.
4  print(height)   # 1.68
5  print(pi)       # 3.14
6  print(weight)   # 68.0
```

Логический тип данных

Тип **bool** представляет два логических значения: **True** (верно, истина) или **False** (неверно, ложь). Значение **True** служит для того, чтобы показать, что что-то истинно. Тогда как значение **False**, наоборот, показывает, что что-то ложно. Пример переменных данного типа:

```
1  isMarried = False
2  print(isMarried)     # False
3
4  isAlive = True
5  print(isAlive)       # True
```

Строковый тип данных

Тип **str** представляет **строки**. Строка представляет последовательность символов, заключенную в одинарные или двойные кавычки, например "hello" и 'hello'. В Python 3.x строки представляют набор символов в кодировке Unicode

```
1  message = "Hello World!"
2  print(message)  # Hello World!
3
4  name = 'Tom'
5  print(name)  # Tom
```

Примеры строк

При этом если строка имеет много символов, ее можем разбить ее на части и разместить их на разных строках кода. В этом случае вся строка заключается в круглые скобки, а ее отдельные части - в кавычки:

```
1 text = ("Laudate omnes gentes laudate "
2         "Magnificat in secula ")
3 print(text)
```

Примеры строк

Если же мы хотим определить многострочный текст, то такой текст заключается в тройные двойные или одинарные кавычки:

```
'''
Это комментарий
'''
text = '''Laudate omnes gentes laudate
Magnificat in secula
Et anima mea laudate
Magnificat in secula
'''
print(text)
```

Управляющие последовательности для строк

Строка может содержать ряд специальных символов - **управляющих последовательностей**. Некоторые из них:

**** - позволяет добавить внутрь строки слеш
\' - позволяет добавить внутрь строки одинарную кавычку
\" - позволяет добавить внутрь строки двойную кавычку
\n - осуществляет переход на новую строку
\t - добавляет табуляцию (4 отступа)

Применим несколько последовательностей:

```
1 text = "Message:\n\"Hello World\""
2 print(text)
```

```
Message:
"Hello World"
```

Пример

Хотя подобные последовательности могут нам помочь в некоторых задачах, однако они также могут и мешать. Например:

```
1  path = "C:\python\name.txt"
2  print(path)
```

Здесь переменная path содержит некоторый путь к файлу. Однако внутри строки встречаются символы "\n", которые будут интерпретированы как управляющая последовательность. Так, мы получим следующий консольный вывод:

```
C:\python
ame.txt
```

Чтобы избежать подобной ситуации, перед строкой ставится символ **r**

```
1  path = r"C:\python\name.txt"
2  print(path)
```

Индексы

name = ‘Маруся’
«М», «а», «р», «у», «с», «я» – элементы строки.
Индексация всегда начинается с нуля.

Зная индекс элемента, можно вывести на экран консоли определенный символ из строки:

print(name[5])

Индексы

Создайте файл и в нем пропишите переменную name = ‘Маруся’.

С помощью встроенной функции print(), выведите определенные элементы строки таким образом, чтобы в консольном окне появилось имя «Муся», не изменяя при этом саму переменную name.

Индексы

```
name = «Маруся»
print(name[0]+name[3]+name[4]+name[5])
```

Индексы

Изменить код таким образом, чтобы в консольном окне отобразилось:
«я саМ»

Индексы

```
name = 'Маруся'
print(f'{name[5]}  {name[4]}{name[1]}{name[0]}')
```

Встраивание в строку значения других переменных

Python позволяет встраивать в строку значения других переменных. Для этого внутри строки переменные размещаются в фигурных скобках **{}**, а перед всей строкой ставится символ **f**:

```
1 userName = "Tom"
2 userAge = 37
3 user = f"name: {userName}  age: {userAge}"
4 print(user)   # name: Tom  age: 37
```

В данном случае на место {userName} будет вставляться значение переменной userName. Аналогично на место {userAge} будет вставляться значение переменной userAge.

Динамическая типизция

Python является языком с **динамической типизацией**. А это значит, что переменная не привязана жестко с определенному типу.

Тип переменной определяется исходя из значения, которое ей присвоено. Так, при присвоении строки в двойных или одинарных кавычках переменная имеет тип **str**. При присвоении целого числа Python автоматически определяет тип переменной как int. Чтобы определить переменную как объект **float**, ей присваивается дробное число, в котором разделителем целой и дробной части является точка.

Изменение типа данных

При этом в процессе работы программы мы можем изменить тип переменной, присвоив ей значение другого типа:

```
1  userId = "abc"  # тип str
2  print(userId)
3
4  userId = 234  # тип int
5  print(userId)
```

С помощью встроенной функции **type()** динамически можно узнать текущий тип переменной:

```
1  userId = "abc"       # тип str
2  print(type(userId)) # <class 'str'>
3
4  userId = 234         # тип int
5  print(type(userId)) # <class 'int'>
```

Формула преобразования типа

Для изменения типа данных используется следующая конструкция:

ПЕРЕМЕННАЯ = НУЖНЫЙ_ТИП(ПЕРЕМЕННАЯ)

Пример

```
1    a = "567"   # текущий тип данных - str
2    a = int(a)  # тип данных переменной a сменился на int
3    print(type(a))
```

Консольный вывод:

```
D:\python\poz_a
<class 'int'>
```

Пример преобразования типа данных

Выше было написано, что при вводе через консоль: **a = input()**, Python воспринимает введенную последовательность символов как **str.** Чтобы Python сразу воспринимал ввод как конкретный тип данных, можно написать так:

a = **int(**input()**)** или a = **float(**input()**)**

В этих случаях введенная последовательность символов будет сразу числовым типом данных.

Резюме

- Переменные предназначены для хранения данных.
- **Camel case** подразумевает, что каждое новое подслово в наименовании переменной начинается с большой буквы
- **Underscore notation** подразумевает, что подслова в наименовании переменной разделяются знаком подчеркивания
- Определив переменную, мы можем использовать её в программе
- Значение переменной можно менять в процессе работы программы
- **Тип данных** – описание того, какого рода данные хранятся в переменной
- Переменная хранит данные одного из типов данных:
- **int** (integer) – число
- **float** (плавающая точка) – дробное число
- **str** (string) – строка
- **bool** (булева функция) – True или False (правда или ложь / из двоичной логики)
- Объявлен

int	float	str	Bool
a = 10	a = 3.5	a = “Hello World!”	a = True

Операторы в Python

Python поддерживает все распространенные арифметические операции:

+ сложение
- вычитание
= присваивание
* умножение
/ деление
** возведение в степень
// целочисленное деление (отбрасывание остатка от деления, дробной части)
% остаток от деления (отбрасывание целой части от деления)

Пример операций на двух **целочисленных (int)** переменных:

Операторы в Python

```
a = 10
b = 5
c = a + b   # 15 - int
c = a - b   # 5 - int
c = a       # в переменной с будет храниться 10
c = a * b   # 50 - int
c = a / b   # 2.0 - float
c = a // b  # 2 - int
c = a % b   # 0 - int
```

Операторы для строк

Также возможно сложение строк: **str + str**; и умножение строки на число: **str * int**:

```
1    a = "Hello"
2    b = "World"
3    a = a + b
4    print(a)
5    b = 2
6    a = a * b
7    print(a)
```

Консольный вывод:

```
HelloWorld
HelloWorldHelloWorld

Process finished with exit code 0
```

Порядок действий

Операции и их направление

** - Справа налево

* / // % - Слева направо

+ - - Слева направо

```
1 number = 3 + 4 * 5 ** 2 + 7
2 print(number)  # 110
```

Здесь начале выполняется возведение в степень (5 ** 2) как операция с большим приоритетом, далее результат умножается на 4 (25 * 4), затем происходит сложение (3 + 100) и далее опять идет сложение (103 + 7).

Чтобы переопределить порядок операций, можно использовать скобки:

```
1 number = (3 + 4) * (5 ** 2 + 7)
2 print(number)  # 224
```

Арифметические операции с присвоением

Ряд специальных операций позволяют использовать присвоить результат операции первому операнду:

```
1  a = 10
2  b = 5
3  a += b    # a = a + b
4  a -= b    # a = a - b
5  c = a        # в переменной c будет храниться 10
6  a *= b    # a = a * b
7  c /= b    # a = a / b
8  a //= b   # a = a // b
9  a %= b    # a = a % b
```

Подведение итогов

Сегодня мы познакомились с:

- Введением в язык программирования Python, синтаксисом;
- Переменными и типами данных, преобразованием типом данных;
- Базовыми операторами, операциями с присвоением