

Программирование

старший преподаватель

кафедры МКМ, Байтелиева Алтын Адилхановна

Лекция 1.

Введение в программирование в Python.

Основной синтаксис языка программирование Python.

Знакомство с Python

Краткая историческая справка

Язык программирования Python был создан к 1991 году голландцем Гвидо ван Россумом.

Свое имя – Пайтон (или Питон) – получил от названия телесериала, а не пресмыкающегося.

После того, как Россум разработал язык, он выложил его в Интернет, где сообщество программистов присоединилось к его улучшению.

Основные особенности языка

Python – интерпретируемый язык программирования. Это значит, что исходный код частями преобразуется в машинный в процессе его чтения специальной программой – интерпретатором.

Python характеризуется ясным синтаксисом. Читать код на нем легче, чем на других языках программирования, так как в Питоне мало используются такие вспомогательные синтаксические элементы как скобки, точки с запятыми. С другой стороны, **правила языка заставляют программистов делать отступы** для обозначения вложенных конструкций. Понятно, что хорошо оформленный текст с малым количеством отвлекающих элементов читать и понимать легче.

Переменные и типы данных

Переменные

Переменные предназначены для хранения данных. Название переменной в Python должно начинаться с алфавитного символа или со знака подчеркивания и может содержать алфавитно-цифровые символы и знак подчеркивания. И кроме того, название переменной не должно совпадать с названием ключевых слов языка Python. Ключевых слов не так много, их легко запомнить:

1	False	await	else	import	pass
2	None	break	except	in	raise
3	True	class	finally	is	return
4	and	continue	for	lambda	try
5	as	def	from	nonlocal	while
6	assert	del	global	not	with
7	async	elif	if	or	yield

И также надо учитывать регистрозависимость, поэтому переменные `name` и `Name` будут представлять разные объекты.

```
1 # две разные переменные
2 name = "Tom"
3 Name = "Tom"
```

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

```
1 name = "Tom" # определение переменной name
2 print(name)  # вывод значения переменной name на консоль
```

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

```
1 name = "Tom" # определение переменной name
2 print(name)  # вывод значения переменной name на консоль
```

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

Переменная хранит данные одного из типов данных. В Python существует множество различных типов данных. В данном случае рассмотрим только самые базовые типы: **bool**, **int**, **float**, **complex** и **str**.

Логические значения

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
```

Целые числа

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.

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

```
1 a = 0b11
2 b = 0b1011
3 c = 0b100001
4 print(a)      # 3 в десятичной системе
5 print(b)      # 11 в десятичной системе
6 print(c)      # 33 в десятичной системе
```

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

```
1 a = 0o7
2 b = 0o11
3 c = 0o17
4 print(a)      # 7 в десятичной системе
5 print(b)      # 9 в десятичной системе
6 print(c)      # 15 в десятичной системе
```

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

```
1 a = 0x0A
2 b = 0xFF
3 c = 0xA1
4 print(a)      # 10 в десятичной системе
5 print(b)      # 255 в десятичной системе
6 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.

Дробные числа

Тип **float** представляет число с плавающей точкой, например, 1.2 или 34.76. В качестве 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
```

Число с плавающей точкой можно определять в экспоненциальной записи:

```
1 x = 3.9e3
2 print(x)    # 3900.0
3
4 x = 3.9e-3
5 print(x)    # 0.0039
```

Число float может иметь только 18 значимых символов. Так, в данном случае используются только два символа – 3.9. И если число слишком велико или слишком мало, то мы можем записывать число в подобной нотации, используя экспоненту. Число после экспоненты указывает степень числа 10, на которое надо умножить основное число – 3.9.

Комплексные числа

Тип **complex** представляет комплексные числа в формате вещественная_часть+мнимая_часть**j** - после мнимой части указывается суффикс **j**

```
1 complexNumber = 1+2j
2 print(complexNumber)    # (1+2j)
```

Строки

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
```

Управляющие последовательности в строке

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)
```

Консольный вывод программы:

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

Динамическая типизация

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.

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'>
```

Консольный ввод и вывод

Вывод на консоль

Для вывода информации на консоль предназначена встроенная функция **print()**. При вызове этой функции ей в скобках передается выводимое значение:

Отличительной особенностью этой функции является то, что по умолчанию она выводит значение на отдельной строке. Например:

```
1 print("Hello World")
2 print("Hello METANIT.COM")
3 print("Hello Python")
```

Здесь три вызова функции `print()` выводят некоторое сообщение. Причем при выводе на консоль каждое сообщение будет размещаться на отдельной строке:

```
Hello World
Hello METANIT.COM
Hello Python
```

Такое поведение не всегда удобно. Например, мы хотим, чтобы все значения выводились на одной строке. Для этого нам надо настроить поведение функции с помощью параметра **end**. Этот параметр задает символы, которые добавляются в конце к выводимой строке и `.` При применении параметра **end** вызов функции `print()` выглядит следующим образом:

```
1 | print(значение, end = конечные_символы)
```

По умолчанию `end` равен символу `"\n"`, который задает перевод на следующую строку. Собственно поэтому функция `print` по умолчанию выводит передаваемое ей значение на отдельной строке.

То есть теперь выводимые значения будут разделяться пробелом:

```
Hello World Hello METANIT.COM Hello Python
```

Причем это может быть не один символ, а набор символов:

```
1 print("Hello World", end=" and ")
2 print("Hello METANIT.COM", end=" and ")
3 print("Hello Python")
```

В данном случае выводимые сообщения будут отделяться символами " and ":

```
Hello World and Hello METANIT.COM and Hello Python
```

Консольный ввод

Наряду с выводом на консоль мы можем получать ввод пользователя с консоли, получать вводимые данные. Для этого в Python определена функция **input()**. В эту функцию передается приглашение к вводу. А результат ввода мы можем сохранить в переменную. Например, определим код для ввода пользователем имени:

```
1 name = input("Введите свое имя: ")
2 print(f"Ваше имя: {name}")
```

В данном случае в функцию `input()` передается приглашение к вводу в виде строки "Введите свое имя: ". Результат функции - результат ввода пользователя передается в переменную `name`. Затем мы можем вывести значение этой переменной на консоль с помощью функции `print()`. Пример работы кода:

Введите свое имя: Eugene

Ваше имя: Eugene

Еще пример с вводом нескольких значений:

```
1 name = input("Your name: ")
2 age = input("Your age: ")
3 print(f"Name: {name} Age: {age}")
```

Пример работы программы:

```
Your name: Tom
Your age: 37
Name: Tom Age: 37
```

Стоит учитывать, что все введенные значения рассматриваются как значения типа **str**, то есть строки. И даже если мы вводим число, как в втором случае в коде выше, то Python все равно будет рассматривать введенное значение как строку, а не как число.

Арифметические операции с числами

Python поддерживает все распространенные арифметические операции:

- +

Сложение двух чисел:

```
1 | print(6 + 2) # 8
```

- -

Вычитание двух чисел:

```
1 | print(6 - 2) # 4
```

- *

Умножение двух чисел:

```
1 | print(6 * 2) # 12
```

- /

Деление двух чисел:

```
1 print(6 / 2) # 3.0
```

- //

Целочисленное деление двух чисел:

```
1 print(7 / 2) # 3.5
2 print(7 // 2) # 3
```

Данная операция возвращает целочисленный результат деления, отбрасывая дробную часть

- **

Возведение в степень:

```
1 print(6 ** 2) # Возводим число 6 в степень 2. Результат - 36
```

- %

Получение остатка от деления:

```
1 print(7 % 2) # Получение остатка от деления числа 7 на 2. Результат - 1
```

При последовательном использовании нескольких арифметических операций их выполнение производится в соответствии с их приоритетом. В начале выполняются операции с большим приоритетом. Приоритеты операций в порядке убывания приведены в следующей таблице.

Операции Направление

** Справа налево

* / // % Слева направо

+ - Слева направо

Пусть у нас выполняется следующее выражение:

```
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
```

Следует отметить, что в арифметических операциях могут принимать участие как целые, так и дробные числа. Если в одной операции участвует целое число (int) и число с плавающей точкой (float), то целое число приводится к типу float.

Арифметические операции с присвоением

Ряд специальных операций позволяют использовать присвоить результат операции первому операнду:

- **`+=`**

Присвоение результата сложения

- **`-=`**

Присвоение результата вычитания

- **`*=`**

Присвоение результата умножения

- **/=**

Присвоение результата от деления

- **//=**

Присвоение результата целочисленного деления

- ****=**

Присвоение степени числа

- **%=**

Присвоение остатка от деления

Примеры операций:

```
1 number = 10
2 number += 5
3 print(number)    # 15
4
5 number -= 3
6 print(number)    # 12
7
8 number *= 4
9 print(number)    # 48
```

Округление и функция round

При операциях с числами типа float надо учитывать, что результат операций с ними может быть не совсем точным. Например:

```
1 first_number = 2.0001
2 second_number = 5
3 third_number = first_number / second_number
4 print(third_number) # 0.40002000000000004
```

В данном случае мы ожидаем получить число 0.40002, однако в конце через ряд нулей появляется еще какая-то четверка. Или еще одно выражение:

```
1 print(2.0001 + 0.1) # 2.1001000000000003
```

В случае выше для округления результата мы можем использовать встроенную функцию **round()**:

```
1 first_number = 2.0001
2 second_number = 0.1
3 third_number = first_number + second_number
4 print(round(third_number)) # 2
```

В функцию `round()` передается число, которое надо округлить. Если в функцию передается одно число, как в примере выше, то оно округляется до целого.

Функция `round()` также может принимать второе число, которое указывает, сколько знаков после запятой должно содержать получаемое число:

```
1 first_number = 2.0001
2 second_number = 0.1
3 third_number = first_number + second_number
4 print(round(third_number, 4)) # 2.1001
```

В данном случае число `third_number` округляется до 4 знаков после запятой.

Если в функцию передается только одно значение - только округляемое число, оно округляется то ближайшего целого

Примеры округлений:

```
1 # округление до целого числа
2 print(round(2.49)) # 2 - округление до ближайшего целого 2
3 print(round(2.51)) # 3
```

Однако если округляемая часть равна одинаково удалена от двух целых чисел, то округление идет к ближайшему четному:

```
1 print(round(2.5)) # 2 - ближайшее четное
2 print(round(3.5)) # 4 - ближайшее четное
```

Округление производится до ближайшего кратного 10 в степени минус округляемая часть:

```
1 # округление до двух знаков после запятой
2 print(round(2.554, 2)) # 2.55
3 print(round(2.5551, 2)) # 2.56
4 print(round(2.554999, 2)) # 2.55
5 print(round(2.499, 2)) # 2.5
```

Спасибо за внимание!