Введение в программирование на Python

Цель занятия

После освоения темы вы:

- узнаете принципы организации кода;
- установите интерпретатор Python на компьютер;
- установите среду разработки PyCharm;
- напишите простой код на Python;
- узнаете базовые типы и конструкции языка Python;
- используете базовые типы и конструкции Python для написания простых программ.

План занятия

1. О языке Python
2. Работа в IDE PyCharm. Первая программа
3. Введение в Python. Ввод и вывод данных
4. Типы данных и операции над ними
5. Условный оператор
6. Цикл `while`
7. Цикл `for`
8. Операторы `continue` и `break`

Используемые термины

Компьютерная программа (далее — программа) — совокупность команд на понятном компьютеру языке для выполнения некоторых функций.

Язык программирования — набор правил для написания программы.

Интерпретатор — программа, которая выполняет код, написанный на языке программирования, например, интерпретатор Python позволяет писать программу в интерактивном режиме.

Инструкция — команда, определяющая, какие операции программа выполнит над данными.

Переменная — именованная область оперативной памяти, в которой хранятся некоторые данные определённого типа.

Тип данных — характеристика, которая определяет возможные значения переменных и действия над ними.

Цикл — конструкция языка программирования, используемая при повторении одних и тех же команд несколько раз.

Конспект занятия

1. О языке Python

Python — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода.

Python является одним из первых языков, в которых упор сделан на его простоту: язык построен таким образом, чтобы программный код был удобно читать.

Для более подробного ознакомления с Python существуют два сайта:

- официальный сайт (https://www.python.org/);
- документация (https://docs.python.org/3/).

В рейтингах языков программирования Python занимает одно из первых мест (например, в рейтинге TIOBE Programming Community).

Python поддерживает все современные парадигмы программирования:

- структурное;
- объектно-ориентированное (ООП);
- функциональное;
- императивное;
- аспектно-ориентированное.

Основные архитектурные черты языка:

- динамическая сильная неявная типизация;
- автоматическое управление памятью;
- полная интроспекция;
- механизм обработки исключений;
- поддержка многопоточных вычислений;

- удобные высокоуровневые структуры данных.

Запуск программ происходит следующим образом (рисунок 1): код записывается в среде разработки, затем с помощью интерпретатора выполняется запуск программы на компьютере. Таким образом, интерпретатор представляет собой переводчик между языком программирования и компьютером. После обработки в интерпретаторе программа выполняется на компьютере.

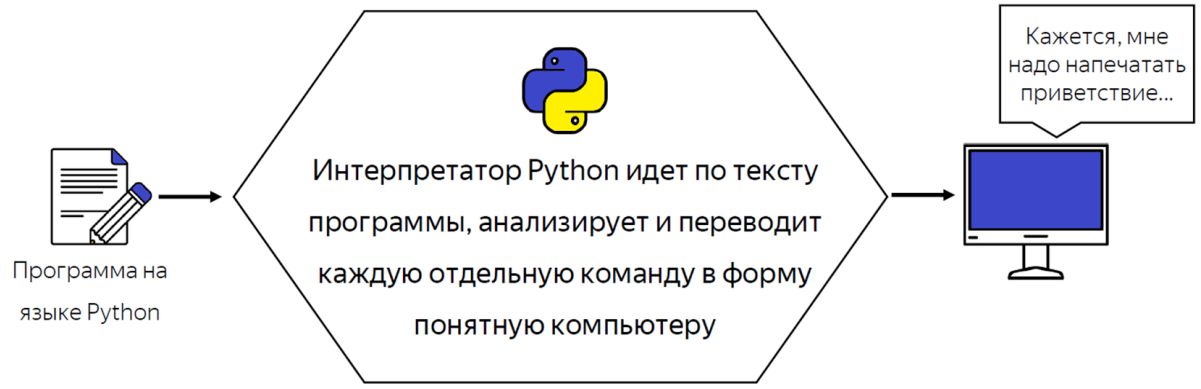

Рисунок 1. Интерпретатор Python.

Особенности Python:

- Python — самый стильный язык программирования в мире;
- Python не допускает двоякого написания кода;
- в Python код можно написать только одним способом;
- в Python отсутствуют лишние конструкции;
- более читаемого кода нет ни в одном другом языке программирования.

Язык программирования Python достаточно молодой, на сентябрь 2022 года актуальной является версия 3.10. До этого существовали версии 1.X и 2.X, которые не поддерживаются. В Python не сформировалась обратная совместимость версий языка. То есть, если программа написана на языке версии 2, она может не запуститься на интерпретаторе версии 3. И наоборот.

Язык применяют во всей индустрии ИТ и во многих ИТ-гигантах: Яндекс, VK, Google и другие. В образовании Python получил наибольшее распространение в США, Python изучают в школах и университетах. Часто Python рекомендуют как первый язык программирования.

Плюсы языка Python:

- открытый код,
- простота синтаксиса,

- наличие мощных библиотек,
- многоплатформенность,
- бесплатность,
- проста в изучении,
- многочисленное сообщество поддержки.

Минусы языка Python:

- невысокая скорость выполнения программ,
- медленная эволюция языка.

Для написания программы удобно использовать интегрированную среду разработки — IDE (англ. Integrated Development Environment). Примеры IDE для написания кода на Python:

- IDLE
- PyDev
- Aptana
- Wing
- PyCharm
- Eclipse
- Emacs
- NotePad++

2. Работа в IDE PyCharm. Первая программа

PyCharm — среда, используемая для написания программ на Python. В IDE есть инструменты для анализа кода, графическая отладчик, встроенное модульное тестирование и поддержка веб-разработки.

Для установки PyCharm сначала нужно установить интерпретатор Python.

Чтобы установить IDE PyCharm перейдите на сайт среды разработки. Далее на главной странице по нажатию на кнопку «*Download*» перейдите на страницу загрузки PyCharm.

Дальше необходимо выбрать версию PyCharm:

- Professional — платная версия с полным набором функций. Она идеально подходит для профессиональной разработки.
- Community — бесплатная версия, ею можно пользоваться благодаря набору базовых возможностей. Для курса будет достаточно возможностей версии Community.

После нажатия на кнопку «*Скачать*» скачивание начнется автоматически.

Далее необходимо запустить установку, в окне установщика нажмите кнопку «*Next*».

При установке есть возможность выбрать папку для установки PyCharm, можно оставить папку установки по умолчанию.

В окне параметров установки поставьте галочки напротив пунктов:

- *PyCharm Community Edition*
- *Add "Open Folder as Project"*
- *Add "bin" folder to the PATH*
- *.py*

Далее запускаем установку, нажав кнопку «*Install*». В конце установки программа может попросить перезагрузить компьютер.

После запуска PyCharm, если вы только что установили IDE и запускаете ее в первый раз, может высветиться окно с предложением установки или импорта предыдущей конфигурации. Можно оставить настройки в появившемся окне по умолчанию.

При первом запуске PyCharm есть возможность открыть существующий проект или создать новый. Поскольку мы только начинаем изучение Python, выбираем создание нового проекта. Затем выбираем директорию для сохранения проекта и нажимаем кнопку «*Create*».

После выполнения указанных действий откроется окно для написания программы.

Далее можно настроить среду разработки, например, сменить тему. Для этого переходим в главном меню: *File→Settings*. В появившемся окне на вкладке Appearance изменим тему, например, для светлой темы можно выбрать *IntelliJ Light*.

При создании нового проекта в окне PyCharm создается файл `main.py`, в котором уже содержится некоторая информация: быстрые клавиши и простейшая программа.

Важно! В правом нижнем углу окна программы PyCharm должно появится название интерпретатора с указанием его версии. Наличие указанной строки означает, что вы сделали все верно, среда разработки видит и может работать с интерпретатором Python. Если название интерпретатора отсутствует, необходимо нажать мышью на область, где должна быть записана строка с информацией о нем. Так вы перейдете в настройки. Далее в настройках укажите интерпретатор вручную. Также возможна полная переустановка интерпретатора и среды разработки.

Используйте этот код, чтобы проверить работу PyCharm:

```
print("Hello World!")
```

Для запуска написанной программы в любом месте кликните правой кнопкой мыши и выберите «*Run*». Программа выполнится, а вывод будет показан на экране. Это значит, что PyCharm установлен и готов к работе.

3. Введение в Python. Ввод и вывод данных

Программа (проект) на языке Python представляет собой несколько модулей — отдельных файлов, содержащих код на языке Python.

Сам **код** — набор инструкций, то есть последовательность определенных команд, которые выполняются интерпретатором. Самые простые программы содержат один модуль. Более сложные программы содержат несколько модулей. Модули могут быть связаны между собой, а могут выполнять не зависящие друг от друга функции.

То есть **модуль** — ряд связанных между собой операций.

Инструкция — указание компьютеру, определяющее, какие операции компьютер выполнит над данными.

Инструкции (или команды) могут быть простые и составные:

- простые состоят из одной строки кода,
- составные содержат вложенные инструкции.

Выражения в составе инструкций определяют, над какими именно данными будут выполнены действия, описанные в инструкции.

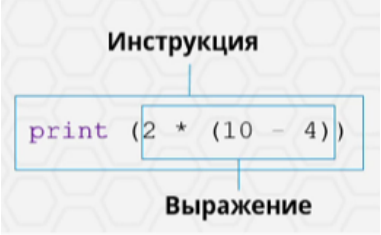

Рисунок 2. Пример команды, содержащей инструкцию.

Операции — любые действия над операндами.

Операнды — некоторые данные:

- литералы,
- выражения,
- переменные.

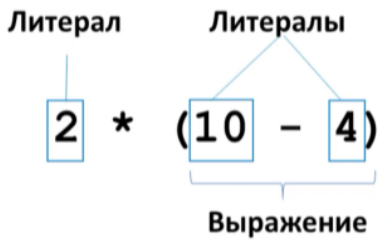

Рисунок 3. Пример литералов в выражении.

Важно! Приоритет выполнения операций соответствует принятому в математике.

Переменная — именованная область оперативной памяти, в которой хранятся некоторые данные определённого типа. Тип данных может быть: целые и вещественные числа, строки и другие.

При определении переменной резервируется ячейка оперативной памяти, и в нее записывается значение переменной. Пример присвоения значения переменной:

```
a = 25
```

Важно! При задании имени переменной можно использовать только ряд допустимых символов:

- заглавные и строчные буквы английского алфавита (A–Z, a–z),
- цифры (0–9),
- знак подчеркивания (_).

Не допускается имя переменной, начинающееся с цифры. Также имена переменных чувствительны к регистру: `name` и `Name` — это две разные переменные.

В Python есть «красивый» способ обмена переменных значениями, который отличает его от других языков программирования:

```
a, b = b, a
```

В результате такого кода переменные обменяются своими значениями.

Особенности синтаксиса языка Python:

- не содержит операторных скобок (begin..end в Pascal или {..} в C),
- блоки выделяются отступами: пробелами или табуляцией, а вход в блок из операторов осуществляется двоеточием,
- однострочные комментарии начинаются со знака фунта `#`, многострочные — начинаются и заканчиваются тремя двойными кавычками `"""`,
- чтобы присвоить значение переменной используется знак `=`, а для сравнения — `==` (два знака равенства),
- для увеличения значения переменной, или добавления к строке используется оператор `+=` (знаки плюс и равно), а для уменьшения — `-=` (знаки минус и равно).

Ввод и вывод данных рассмотрим на примере задачи о сложении двух чисел: пользователь вводит в консоль два числа, компьютер сам считает их сумму, результат также выводится в консоль.

Для ввода значения переменной с клавиатуры используется функция `input()`. То есть программа ждет, пока пользователь введет значение и нажмет *Enter*. После чего введенное значение записывается в переменную, или как говорят, связывается с именем переменной.

Важно! Результат функции `input()` — строка символов вне зависимости от содержимого ввода. Чтобы преобразовать вводимые с клавиатуры символы в целое число, надо указать тип вводимой переменной: `a = int(input())`.

При использовании функции `input()` возможно использование подсказки:

```
a = int(input("Введите число: "))
```

Для вывода значения переменной используется функция `print()`:

```
print(a) #значение переменной
print("Ответ:", a) #значение и текст
```

Возможно более сложное использование функции `print()` — одновременный вывод текста и нескольких переменных:

```
print("Ответ:", a+b)
print(a, "+", b , "=", c)
```

Параметр `sep` в функции `print()` позволяет настроить работу разделителей. Например, чтобы «очистить» вывод от пробелов используйте команду:

```
print(a, "+", b, "=", c, sep="")
```

Пример программы сложения двух чисел.

```
a = int(input())
b = int(input())
c = a + b
print(c)
```

Важно! Результат вычислений суммы записан в переменную `c`, то есть под результат выделена отдельная ячейка памяти. В дальнейшем результат сложения (переменная `c`) может использоваться в программе.

Пример программы сложения двух чисел (более удобный для пользователя вариант):

```
print("Введите два числа:")
a = int(input())
```

```
b = int(input())
c = a + b
print(a, "+", b , "=", c, sep="")
```

Команды для ввода и вывода данных:

- ввод данных (пользователь должен что-то напечатать на клавиатуре) — `input()`,
- вывод данных (программа должна что-то показать на экране) — `print()`.

Важно! Избегайте использования пробелов сразу перед открывающей скобкой, после которой начинается список аргументов функции.

- Правильно: `print("Привет!")`
- Неправильно: `print  ("Привет!")`

В среде разработки PyCharm код программы записывается в верхнем окне, а консоль для ввода-вывода располагается в нижней части. Также в консоль выводится сообщения об ошибках.

Напишем простейшие программы на языке Python: программу “повторения” и программу “приветствие пользователя”.

Программа «повторения» сводится к повтору введенного пользователем текста.

Для того, чтобы программа считала строку с консоли используем команду `input()`. Запишем результат в переменную

```
t = input()
```

Так как в команде `input()` можно использовать подсказку, дополним ее

```
t = input("Введите текст")
```

Для вывода используем команду `print()`

```
print(t)
```

Мы можем изменить программу, сделав чтобы введенный текст повторялся дважды. В этом случае нужно еще раз повторить команду print(t).

Текст программы «повторения» полностью:

```
t = input("Введите текст")
print(t)
print(t)
```

Программа «приветствие пользователя» запрашивает имя пользователя, а затем приветствует пользователя по введенному имени.

Будем использовать для вывода команду

```
print("Как вас зовут?")
```

После вывода запрашиваем имя и записываем его в переменную `Name`

```
Name = input()
```

Снова используем команду, на этот раз выведем имя

```
print("Рады Вас видеть,", Name, "!")
```

Текст программы «приветствие пользователя» полностью:

```
print("Как вас зовут?")
Name = input()
print("Рады Вас видеть,", Name, "!")
```

4. Типы данных и операции над ними

На рисунке 1 показаны основные простые типы данных в Python.

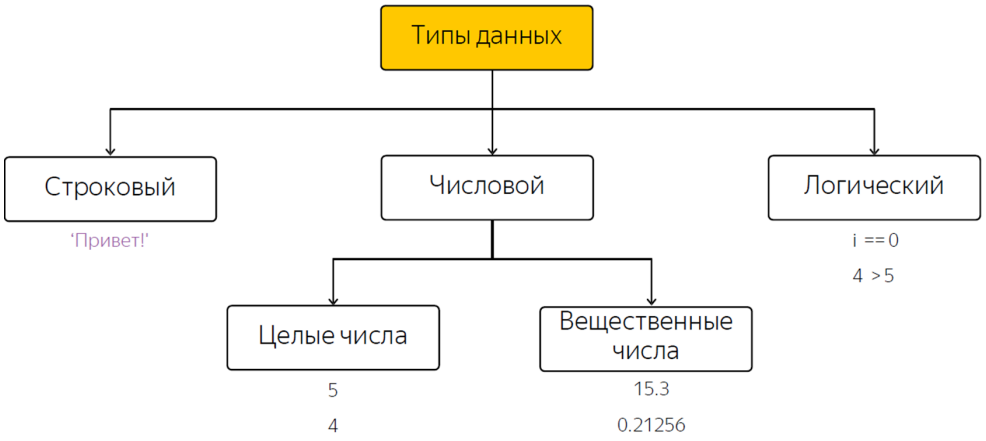

Рисунок 1. Простые типы данных

Числовый тип данных.

Целые числа (integer) — положительные и отрицательные целые числа, а также 0 (например: 4, 687, −45, 0). Для преобразование в целое число используется команда `int()`.

Числа с плавающей точкой (float point) — дробные числа (например: 1.45, -3.789654, 0.00453). Разделителем целой и дробной части служит символ точки «.». Для преобразование в целое число используется команда `float()`.

Не всегда при выполнении математических операций тип данных результата совпадает с типом данных операндов. Например, при делении целых чисел результатом будет вещественное число. В таблице 1 показаны основные математические операторы, их описание и примеры использования.

Таблица 1. Математические операторы.

Оператор	Описание	Пример	Результат
+	Сложение	7 + 3	10
–	Вычитание	7 – 3	4
*	Умножение	7 * 3	21
/	Деление (истинное)	7/3	2.3333333333333335
**	Возведение в степень	7 ** 3	343
//	Целочисленное деление	7 // 3	2
%	Остаток от деления	7 % 3	1

Нужно помнить о приоритете выполняемых операций:

1. Возведение в степень(`**`)
2. Унарный минус (`-`)
3. Умножение (`*`), деление(`/ % //`)
4. Сложение (`+`), вычитание (`-`)
5. Операторы сравнения (`<= <> >=`)
6. Операторы равенства (`== !=`)
7. Логические операторы (`not or and`)
8. Оператор присваивания (`=`)

Если одна из операций в выражении не совпадает со списком выше, нужно поместить ее в скобки.

В Python есть возможность поменять тип данных с помощью функций:

- `int()` — преобразует аргумент в целое число
- `float()` — преобразует аргумент в число с плавающей точкой

Пример кода, преобразующего введенную пользователем строку в число и записывает в переменную `a`

```
a = int(input())
```

Пример программы сложения двух целых чисел: в программе считываются два числа с консоли, преобразуются в целые числа, затем выводится их сумма с помощью функции `print()`.

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

Пример программы сложения двух чисел с плавающей точкой: отличие — использование команды `float()` для преобразования типов введенных данных.

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

Программа для нахождения длины окружности, площади круга и объема шара одного и того же радиуса.

Входным параметром является радиус, все остальные параметры рассчитывает программа.

Для ввода значения радиуса используем команду `input()` с подсказкой. Но команда `input()` считывает данные в строковом формате, поэтому используем перевод в вещественное значение — функция `float()`. Значение радиуса считывается в переменную `r`.

```
r = float(input("Введите радиус: "))
```

Точное значение числа π можно считать из специальной библиотеки, но пока мы укажем его в явном виде:

```
pi = 3.14
```

Длину окружности, площади круга и объем шара будем находить по известным математическим формулам:

```
l = 2 * pi * r
s = pi * r ** 2
v = (4 / 3) * pi * r **3
```

Символ «`**`» означает возведение в степень.

Для вывода в консоль результатов расчетов используем команду `print()`:

```
print("l = ", l)
print("s = ", s)
```

```
print("v = ", v)
```

Программа, демонстрирующая операции целочисленного деления и нахождения остатка от деления.

```
a = int(input("Введите делимое: "))
b = int(input("Введите делитель: "))
d = a // b
ost = a % b
print("Результаты целочисленного деления: ", d)
print("Результат остатка от деления:", ost)
```

Программа ждет ввода от пользователя двух целых чисел. Важно использовать именно целые числа, поскольку операции целочисленного деления (`//`) и нахождения остатка от деления (`%`) выполняются только над целыми числами. С помощью команды `print()` выполняется вывод результатов.

Программа для нахождения периметра и площади треугольника по длинам его сторон.

Программа рассчитывает площадь по формуле Герона, все необходимые пояснения приведены в комментариях (`#`).

```
a = int(input("Введите a: "))
b = int(input("Введите b: "))
c = int(input("Введите c: "))
p = a + b + c #расчет периметра
pp = p / 2 #расчет полупериметра
s = (pp * (pp - a) * (pp - b) * (pp - c)) ** 0.5 #расчет площади по формуле Герона
print("Периметр треугольника p = ", p)
print("Площадь треугольника s = ", s)
```

Строковый тип данных.

Строки (string) — набор символов, заключенных в кавычки (например: "ball", "What is your name?", 'dkfjUUv', '6589'). Символами являются буквы, цифры, специальные знаки и пробел. Для преобразование в целое число используется команда `str()`.

Кавычки в Python могут быть одинарными или двойными.

Строки могут использовать спецсимволы, которые не выводятся, но выполняют определенные действия. Например, таким образом, можно вывести символы, которые используются для написания команд.

- `\n` — конец строки
- `\t` — табуляция
- `\'` — одинарная кавычка
- `\\` — обратный слэш

Пример программы для использования спецсимволов.

```
print("восход\t07:15\nзакат\t22:03")
```

В результате будет выведены значения в столбик, так как используется символ табуляции.

Простейшие операции над строками:

- конкатенация (склеивание), несмотря на то, что в примере переменные содержат числа, программа будет работать с ними как со строками, поскольку значения взяты в кавычки.

  ```
  x = '10'
  y = '20'
  print(x + y)
  ```

  Результат работы программы: 1020.

- дублирование позволяет несколько раз повторить строку.

  ```
  x = '10'
  y = '20'
  print(x * 2 + y * 3)
  ```

  Результат работы программы: 1010202020.

Таблица 2. Операции над разными типами данных

Выражение	Результат выполнения, тип данных
34.907 + 320.65	355.55699999999996 число с плавающей запятой
`'Hi, ' + 'world :) '`	'Hi, world :) ' строка, выполняется конкатенация

`'Hi, ' * 10`	'Hi, Hi, Hi, Hi, Hi, Hi, Hi, Hi, Hi, Hi, ' строка, выполняется дублирование
`'Hi, ' + 15`	Ошибка, попытка сложить строку и целое число

Логический тип данных

Логический тип данных необходим, когда требуется проверить на истинность некоторое выражение. Чаще выполняется сравнение. Переменные логического типа могут принимать значения: True (истина) либо False (ложь).

Например:

`2 + 2 == 4` — `True`

`2 + 2 == 5` — `False`

Логический тип данных имеет специальные операции:

- `and` — логическое И, логическое умножение;
- `or` — логическое ИЛИ, логическое сложение;
- `not` — логическое НЕ, логическое отрицание,
- `in` — вхождение левого элемента в то или иное множество, в ту или иную коллекцию.

Таблица 3. Логические операции

Выражение	Результат
`True and True`	`True`
`True and False`	`False`
`False and False`	`False`
`True or True`	`True`
`True or False`	`True`
`False or False`	`False`
`not True`	`False`
`not False`	`True`

5. Условный оператор

Используется, когда некая часть программы должна быть выполнена, только если верно какое-либо условие, либо должна быть выполнена другая часть программы, когда это условие неверно. Может быть, что никакие действия не выполняются.

Для записи используются ключевые слова `if` и `else` («если», «иначе»), двоеточие, а также отступ в четыре пробела.

Синтаксис конструкции языка.

```
if условие:
     действие, если условие верно
else:
     действие, если условие неверно
```

В условии может быть отношение как логическая переменная, также может быть переменная другого типа. Блоки кода определяются через отступы, чаще всего используют четыре пробела или табуляцию.

Пример программы для проверки пароля.

Программа проверяет введенный пароль, если пароль совпадает со строкой `'qwerty'`, выводится сообщение «`Доступ открыт.`», в противном случае — «`Ошибка, доступ закрыт!`».

```
print('Введите пароль')
password = input()
if password == 'qwerty':
    print('Доступ открыт.'
else:
    print('Ошибка, доступ закрыт!')
```

Если в условном операторе требуется проверка нескольких выражений, можно использовать логические операторы:

- `and` — два условия должны выполняться одновременно,
- `or` — достаточно выполнения одного из двух вариантов (или оба сразу),
- `not` — нужно убрать какой-то вариант.

Приоритет логических операций: `not`, `and`, `or`.

Пример использования логической операции and.

В программе условие будет истинным, только при выполнении обоих выражений в операторе `if`.

```
print('Как называются первая и последняя')
print('буквы греческого алфавита?')
greek_letter1 = input()
greek_letter2 = input()
if greek_letter1 == 'альфа' and greek_letter2 == 'омега':
    print('Верно.')
else:
    print('Неверно.')
```

В первых двух строках программы на консоль выводится пояснение, что требуется сделать пользователю. Далее программа считывает строку с помощью команды `input()`. Затем программа выполняет проверку условия с помощью оператора if. Если хотя бы одно из выражений `greek_letter1 == 'альфа' and greek_letter2 == 'омега'` не выполняется, осуществляется переход к блоку `else`.

В блоках `if-else` может использоваться не одна команда, как это рассматривалось в предыдущих примерах, а несколько. Блоки кода в Python выделяются с помощью отступов.

Пример программы, использующей блоки кода. Если условие в операторе if верно, выполняются две команды:

```
print('Аве, Цезарь!')
print('Слава императору!')
```

Если условие неверно, выполняются также две команды, но которые уже прописаны в блоке else:

```
print('Приветик.')
print('Погода сегодня хорошая.')
```

Важно! Количество команд в блоках `if-else` может быть любым, главное — соблюдать отступы.

```
print('Представься, о незнакомец!')
name = input()
if name == 'Цезарь' or name == 'Caesar':
    print('Аве, Цезарь!')
```

```
    print('Слава императору!')
else:
    print('Приветик.')
    print('Погода сегодня хорошая.')
print('Засим - заканчиваем.')
```

В Python допускается конструкция `if-elif`. Эта конструкция используется при проверке нескольких условий.

```
if условие1:
    …
elif условие2:
    …
elif условие3:
    …
```

Цепочка может быть заменена цепочкой `else-if`, как это реализовано в других языках программирования. Но использования формы `if-elif` является в Python предпочтительной.

Пример программы «Личностный тест» с использованием условного оператора в конструкции `if-elif` и блоков кода.

С помощью команды `print()` выводим в консоль два вопроса. Ответы, введенные в консоль, считываем командой input() и записываем в переменные `answer1` и `answer2`:

```
print('Любите ли вы котиков?')
answer1 = input()
print('Умеете ли вы программировать?')
answer2 = input()
```

Рассмотрим все возможные случаи в предположении того, что ответ должен состоять из «да» или «нет»:

- да–да,
- да–нет,
- нет–да,
- нет–нет.

В каждом случае будем выдавать соответствующее сообщение. В программе предусмотрен случай, когда ни один из вариантов не сработал — блок `else`.

```
if answer1 == 'да' and answer2 == 'да':
    print('Да вы просто идеал!')
elif answer1 == 'да' and answer2 == 'нет':
    print('Вы обладаете редкостной добротой.')
elif answer1 == 'нет' and answer2 == 'да':
    print('Вы обладаете незаурядным умом.')
elif answer1 == 'нет' and answer2 == 'нет':
    print('У вас большие перспективы.')
else:
    print('Ошибка: ожидались ответы да/нет')
```

В условном операторе может использоваться оператор `in`, который проверяет содержит ли данная строка конкретную подстроку

`'хорош' in 'хорошо'` — верно

`'хорош' in 'эх, хорошо'` — верно

`'хорош' in 'плохо'` — неверно

`'хорошо!' in 'хорошо'` — неверно

Пример программы проверки наличия подстроки в строке. Программа проверяет два условия, и в зависимости от значения переменной text выводит то или иное сообщение, если ни одно условие неверно, выполняется код в блоке `else`.

```
text = input()
if 'хорош' in text and 'плох' not in text:
    print('Положительная эмоциональная окраска')
elif 'плох' in text and 'хорош' not in text:
    print('Отрицательная эмоциональная окраска')
else:
    print('Нейтральная или смешанная эмоциональная окраска')
```

6. Цикл `while`

Цикл — конструкция языка программирования, используемая при повторении одних и тех же команд несколько раз. То есть цикл будет выполнять блок кода, пока выполнено условие.

В рассматриваемом примере программа считывает действительное число и записывает его в переменную `number`. Пока переменная `number > 0` будет выполняться тело цикла, состоящее из трех строк. Тело цикла отделено отступом слева. Если условие перестанет выполняться, программа выйдет из цикла и перейдет на последнюю строку `print('Вы ввели отрицательное число или ноль. Всё.')`

Условие в цикле `while` очень похоже на условие в конструкции `if`. Но в отличие от условного оператора, команды, прописанные после входа в цикл, будут повторяться до тех пор, пока будет истинным условие.

```
number= float(input())
while number > 0:
    print('Вы ввели положительное число! Вводите дальше.')
    number= float(input())
    print('Так-так, что тут у нас...')
print('Вы ввели отрицательное число или ноль. Всё.')
```

Цикл while сокращает размер кода. В таблице 4 показано сравнение программ сложения действительных чисел, имитирующих цену товара. Числа вводятся с клавиатуры.

Программу из левого столбца легко сократить, используя цикл `while`. В цикле использовано условие: если товары закончились, необходимо ввести `0`. Программа стала более универсальной — можно суммировать любое количество чисел.

Таблица 4. Сравнение программ с и без использования оператора цикла.

Без использования цикла `while`	С использованием цикла `while`

`total = 0`
`price = float(input())`
`total = total + price`
`price = float(input())`
`total = total + price`
`price = float(input())`
`total = total + price`
`print('Сумма введённых чисел`
	`total = 0`
	`print('Вводите цены; для остановки введите 0.')`
	`price = float(input())`
	`while price != 0:`
	`  total = total + price`
	`  price = float(input())`
	`print('Общая стоимость равна',`

равна', total)	total)

Некоторые программы должны выполняться до тех пор, пока пользователь не ввел какое-то значение, или какую-то строку, или не использовал определенную комбинацию клавиш.

Пример программы «Числа до нуля»: пользователь вводит числа, и пока введённые числа не равны нулю, программа работает, как только происходит иное — завершает работу.

```
height = int(input())
while height != 0:
    height = int(input())
```

Составной оператор присваивания используется во многих языках программирования и является стандартом записи изменения переменной. Конструкция позволяет сократить запись кода.

```
number = int(input()) #например, 5
number = number + 1 #number становится равным 6
print(number)
```

В примере вводится значение переменной `number`, в следующей строке кода ее значение увеличивается на 1.

Таблица 5. Составной оператор присваивания.

x = x + y	x += y
x = x - y	x -= y
x = x * y	x *= y
x = x / y	x /= y
x = x // y	x //= y
x = x % y	x %= y
x = x ** y	x **= y

Во всех случаях (таблица 5) происходит изменение переменной `x` на значение переменной `y`.

Программа проверки ввода пароля до тех пор, пока пароль не был введен верно.

Программа запрашивает у пользователя ввод пароля и записывает его в переменную `password`.

```
print("Введите пароль: ")
password = input()
```

Рассмотрим случай, когда запрашиваемый пароль «12345». Если пароль введен неверно, пользователю выводится соответствующее сообщение и предложение еще раз ввести пароль. Таким образом, цикл будет повторяться до тех пор, пока не будет введен нужный пароль.

```
while password != "12345":
    print("Неверный пароль!")
    print("Введите снова")
    password = input()
```

Если пароль введен верно, выводится сообщение:

```
print("Доступ открыт!")
```

7. Цикл `for`

Цикл `for` используется при известном количестве итераций (повторов) тела цикла.

Рассмотрим пример программы с использованием цикла `for`. С консоли с помощью `input()` вводят количество повторений тела цикла `n`. Конструкция `for` использует функцию `range()`, которая в рассматриваемом примере возвращает последовательность целых чисел, начинающуюся с `0` и оканчивающуюся числом `n-1`.

```
n = int(input())
for i in range(n):
    print(i)
```

Тело цикла выполняется `n` раз. Переменная `i` «пробегает» значения от 0 включительно до `n` не включительно:

- сначала `i` равно 0,
- затем `i` равно 1,

- на последней итерации `i` равно `n-1`.

При `n = 10` переменная `i` будет принимать целые значения от 0 до 9.

Тело цикла состоит из одной команды `print(i)`. Переменная `i` как может использоваться, так может и не использоваться в теле цикла.

Пример программы для расчета суммы целых чисел от `0` до `n-1`.

```
n = int(input())
total = 0
for i in range(n):
    print('Рассматриваем число', i)
    total += i
    print('Промежуточная сумма равна', total)
print('Итоговая сумма всех этих чисел равна', total)
```

В программе вводится число `n`, обнуляется счетчик `total`. Далее при каждой итерации рассматривается определенное число, хранящееся в переменной `i`. После прохождения всех итераций выводится итоговая сумма чисел.

Пример программы нахождения суммы четных чисел на отрезке от `m` до `n`.

При решении задачи будем полагать, что `m` и `n` — целые числа, `m < n`.

Сначала вводим значения `m` и `n` с клавиатуры:

```
m = int(input())
n = int(input())
```

Чтобы посчитать сумму, определяем переменную `s` и задаем ей начальное значение:

```
s = 0
```

Все суммируемые числа, будем проверять на четность внутри цикла. Поскольку функция `range()` исключает последнее число из рассмотрения, мы будем использовать запись `range(m, n + 1)`.

```
for i in range(m, n + 1):
    if i % 2 == 0:
        s += i
print(s)
```

8. Операторы `continue` и `break`

Оператор break немедленно прерывает выполнение цикла `for` или `while`. Использование таких операторов может быть необходимо, если на одной из итераций цикла уже был получен ответ.

Пример программы с использованием оператора `break`. Оператор прерывает весь цикл при значении `i = 3`, и происходит переход к концу цикла.

```
for i in range (10):
    print('Итерация номер', i, 'начинается ...')
    if i == 3:
        print('Ха! Внезапный выход из цикла!')
        break
    print('Итерация номер', i, 'успешно завершена.')
print('Цикл завершён')
```

Оператор break может использоваться для выхода из бесконечного цикла. Выход из цикла будет выполняться при вводе слова «стоп» в консоль.

```
while True:
    word = input()
    if word == 'стоп':
        break
    print('Вы ввели:', word)
print('Конец')
```

Оператор `continue` немедленно прерывает текущую итерацию цикла `for` или `while` и переходит к следующей. Например, это необходимо в случае невозможности выполнении текущей итерации при определенном значении, типичный пример — деление на ноль. Все остальные итерации при этом выполняются.

Пример программы с использованием оператора `continue`. Оператор прерывает итерацию при значении `i = 3`, последующие итерации цикла выполняются.

```
for i in range (10):
    print('Итерация номер', i, 'начинается ...')
    if i == 3:
        print('... но не завершается успешно.')
        continue
    print('Итерация номер', i, 'успешно завершена.')
```

```
print('Цикл завершён')
```

Операторы `break` и `continue` имеют особенность использования — они действуют только на внутренний цикл.

В программе используется вложенный цикл, в котором используется оператор `break`. При запуске программы оператор будет прерывать только внутренний цикл, который определяется переменной `j`. Ко внешнему циклу оператор `break` отношение не будет иметь.

```
n = int(input())
for i in range(1, n + 1):
    for j in range(1, n + 1):
        print(i * j, end='\t')
        if j == 7:
            break
    print()
```

Дополнительные материалы для самостоятельного изучения

1. Официальный сайт Python (англ.)
2. Среда разработки IDE PyCharm (англ.)
3. Интерпретатор, интерактивная оболочка (англ.)
4. More Control Flow Tools — Python 3.11.4 documentation. (англ.)