Лекция № 2

Т. Ф. Хирьянов

Преимущества и недостатки языка программирования Python

Достоинства:

1. *Современность.*
   Встроенно очень много удобных методик программирования.
2. *Универсальность.*
   На этом языке можно программировать любое приложение от скрипта операционной системы до игры на мобильном телефоне.
3. *Богатая стандартная библиотека.*
   В ней предусмотренно огромное количество функций, включая работу с сетями и математическими выражениями.
4. *Кроссплатформенность*
   Интерпретатор Python может работать в любой операционной системе, на компьютерах с разной архитектурой.
5. *Интерпретируемость*
   Одним из следствий интерпретируемости является то, что в переменную можно сохранять данные разных форматов.

   ```
   x = 123
   x = "python"
   ```

   Так 123 – это целое число, а python – строка.

Ссылочная модель данных в Python. Динамическая типизация.

В Python нет операции присваивания. Запись

$$x = 123$$

означает, что объект 123 связывается с ссылкой x. А сама операция является связыванием объекта и ссылки. Кусок кода

$$x = int(x)$$

будет выполняться следующим образом. Сначала выполнится выражение стоящее справа, затем порожденный им объект, сохранится в некоторой области памяти, вообще говоря, отличающейся от того участка, на который указывал x ранее. В заключении ссылка x связывается с этим новым объектом.

Для того чтобы справиться с утечкой памяти старый объект удаляется сборщиком мусора, если на него больше нет ссылок.

При таком подходе тип объекта строго определен, но появляется у ссылки только в момент выполнения программы.

Отличия языков программирования Python 2 и Python 3

При развитии и улучшении языков программирования часто бывает необходимо кардинально изменить концепции привычных вещей. При таком переходе теряется совместимость старых и новых версий языка. Так произошло с Python2 и Python3.

Пример.

Функция input() в этих версиях языка ведет себя по разному. Так в Python2 выражение

$$x = input('5 + 3')$$

вычислит значение суммы и сохранит его в x. В Python3 это же выражение выведет на экран

$$5 + 3$$

а затем считает данные с клавиатуры и сохранит их в x в виде строки. Это бывает очень удобно. Например, можно вывести своеобразное приглашение: "Введите x".

Обмен двух переменных значениями

Допустим, есть две переменные

```
x = 3
x = 7
```

$$x = 3$$

$$x = 7$$

и необходимо произвести обмен их значениями. Эта задача однако не так тривиальна, как кажется. Например, такое решение

```
x = y
y = x
```

является неверным, так как после выполнения $x = y$ теряется ссылка на объект «3» и начинает указывать на «7».

В решении данной задачи удобно воспользоваться аналогией. Так, чтобы поменять содержимое стакана с водой и кружки с молоком друг между другом, можно воспользоваться третьим сосудом.

Этот алгоритм можно реализовать следующим образом.

```
tmp = x
x = y
y = x
```

Теперь нет потери значения x, так как оно предварительно сохранено в переменную tmp. Это классический пример обмена значениями переменных через третью.

Кортеж переменных

В языке Pythonсуществует элегантное решение этой задачи, основанное на использовании специальной структуры данных *кортежа переменных*.

Кортеж синтаксически представляет собой набор данных или переменных разделенных запятыми. Обычно он окружается круглыми скобками. Например,

```
(1 ,2 ,3)
```

(1, 2, 3) является кортежем данных, а

```
(x, y, z)
```

кортежем переменных.

Такая структура имеет множество удобных способов применения. Так можно присвоить кортежу переменных кортеж данных.

```
(x, y, z) = (1, 2, 3)
```

Или обменять значениями две переменные более наглядным способом.

```
(x, y) = (y, x)
```

Более того, используя кортежи есть возможность организовать циклический сдвиг нескольких переменных.

```
(x, y, z) = (z, x, y)
```

Арифметические операции

Большинство арифметических операций синтаксически реализованы в Pythonнативным образом, как во многих других языках программирования. Например, операции сложения, умножения и деления выглядят следующим образом.

```
x + y
x * y
x / y
```

Однако существуют некоторые особенности. Первая из них связанна с операцией целочисленного деления div. Во многих языках программирования она будет выполнена следующим образом.

```
−17 div 10 = −1
```

Однако в соответствии с очевидным уравнением

```
x * 10 + r = −17
```

это приведет к тому, что соответствующий остаток будет отрицательным. Это противоречит принятому в математике положению, что остаток от деления по модулю — неотрицательное число. Для решения это проблемы в приведенном выше уравнении можно уменьшить x на единицу. Тогда целая часть и остаток от деления будут следующими.

```
−17 div 10 = −2
−17 % 10 = 3
```

Цикл с предусловием

Для организации циклического выполнения каких-либо действий в Pythonпредусмотрена конструкция цикла с предусловием. Она начинается с зарезервированного слова *while*. Затем записывается условие, при котором будут выполняться действия, записанные в *тело цикла*. После условия необходимо поставить двоеточие и перейти на новую строчку. Перед каждой следующей строкой тела цикла делается отступ. Для того чтобы выйти из тела цикла необходимо и достаточно перейти на новую строчку и убрать относительный отступ.

При однократном выполнении или *итерации* цикла в начале будет проверено условие и если оно выполнено, то начнется последовательное исполнение команд тела цикла и переход к новой итерации по их окончании.

Существуют также *функции управления циклом*. Например, с помощью функции *break* можно преждевременно выйти из цикла. Как правило это бывает нужно, в случаях проверки дополнительного условия в момент выполнения итерации. Условная конструкция оформляется аналогичным образом: после зарезервированного слова *if* следует само условие, оканчивающееся двоеточием. В случае выполнения условия будут выполнены команды, записанные ниже через отступ.

Также бывает необходимо при выполнении какого-то условия завершить выполнение текущей итерации и перейти к следующей. Для этого используется функция *continue*.

В качестве примера оформления можно рассмотреть псевдокод, реализующий бег человека на стадионе.

```
while sunny:
    if trouble:
        break
    run 100 metres
    if difficult:
        continue
    run extremelly last 100 metres
```

При таком алгоритме действий человек будет бегать, пока на улице стоит хорошая погода. В случае форс-мажора, например, травмы, бег прекращается. Но если ничего не произошло, то спортсмен будет бежать трусцой 100 метров. Затем, если он устал, снова проверит погоду, оценит, все ли в порядке, пробежит еще 100 метров трусцой, т. е. перейдет к следующей итерации цикла. Если же человек полон сил, то последние 100 метров круга он пробежит так быстро, как сможет.

Примером настоящего кода может служить следующая программа.

```
x = int(input())
while x < 100:
    s += x
    if x % 7 == 0:
        break
    x += 5
else:
    print(x)
print(s)
```

Она выполняет подсчет суммы арифметической прогрессии с разностью, равной 5, начиная с введенного пользователем числа (которое приводится к целочисленному типу). Подсчет суммы производится пока члены прогрессии меньше 100. Строка

```
s += x
```

означает, что значение переменной s увеличивается на x. При этом если текущий просуммированный член прогрессии делится нацело на 7 (т.е. его остаток при делении на 7 равен 0), то осуществляется выход из цикла. Для сравнения равенства двух чисел используется оператор ==. В завершении программы на экран выводится значение суммы.

Для того чтобы по окончании работы цикла было понятно, завершился ли он нормально или преждевременно, существует специальное дополнение к конструкции цикла. После тела на уроне while записывается else и ставится двоеточие. Инструкции записанные ниже через отступ будут выполнены только когда будет нарушено условие в заголовке цикла, т.е. когда последний завершится нормально. В приведенном примере на экран выведется последнее значение x. Функция break осуществляет полный выход из цикла, а значит, «команды else» будут также не выполнены.

В условной конструкции также есть дополнение else. Синтаксически она выглядит следующим образом.

if условие:
действие A
else:
действие B

Генерация последовательностей.

Генерация арифметической прогрессии от нуля до вводимого пользователем N, не включая последнего, с разностью, равной 1.

```
N = int(input())
i = 0
while i < N:
    print(i)
    i += 1
```

Удобно использовать понятие *инварианта цикла*. Это некоторое утверждение, которое является верным в начале каждой итерации цикла. В данном примере оно может быть следующим: «В момент начала каждой итерации распечатаны все целые числа от 0 до N, не включая последнее».

Вложенные циклы.

Задача вывода на экран показаний часов с 5 до 12, когда минуты равны 10, 20, 30, 40 и 50.

```
hour = 5
while hour <= 12:
    minute = 10
    while minute <= 50:
        print(hour, minute, sep=':')
        minute += 10
    hour += 1
```

Чтобы не прописывать распечатку пяти показаний для каждого часа, логично использовать еще один, вложенный, цикл, в котором итератор будет пробегаться по значениям минут.

Стоит отметить, что функция break осуществляет выход только из цикла, в чьем теле она записана, т.е. с ее помощью нельзя выйти из вложенного цикла.

Функции

Для того чтобы программа была понятной любому программисту, в том числе и ее автору некоторое время спустя, важно давать функциям и переменным содержательные названия. Например, нельзя понять вне контекста результат работы функции f(a, b). Однако можно ожидать, что функция max(a, b) возвратит максимальное из двух чисел a и b.

Рассмотрим реализацию последней в качестве примера синтаксиса.

```
def max(a, b):
    if a > b:
        return a
    return b
```

Заголовок объявления функции начинается с зарезервированного слова *def*. Ввиду того, что Pythonявляется интерпретируемым языком, типы данных аргументов функции не указываются,

так как они определяются только в момент вызова функции. Поэтому приведенная программа будет одинаково хорошо работать как с целыми, так и с вещественными числами.

В данном примере не используется конструкция с else, так как функция return возвращает результат работы функции (значение переменной a или b) и завершает ее выполнение. Поэтому выполнится либо return a, либо return b.

Также функция имеет возможность возвращать кортеж. Примером может служить функция, осуществляющая обмен двух переменных значениями.

```
def swap(x, y):
    return y, x
```