Введение. Язык программирования Python

В рамках данного курса мы будем изучать программирование на примере современного языка программирования – Python 3. Его основные достоинства:
1. Кроссплатформенность и бесплатность.
2. Простой синтаксис и богатые возможности позволяют записывать программы очень кратко, но в то же время понятно.
3. По простоте освоения язык сравним с бейсиком, но куда более богат возможностями и значительно более современен.
4. Богатая стандартная библиотека, возможность разработки промышленных приложений (для работы с сетью, GUI, базами данных и т.д.)

Для программирования на языке Python вам понадобится:

1. Консольный интерпретатор языка.
2. Среда разработки (которая включает в себя текстовый редактор).

Для установки консольного интерпретатора языка Python Последний дистрибутив версии 3.5.2 можно скачать со страницы
https://www.python.org/downloads/release/python-352/
Между различными версиями языка Python есть существенные различия, поэтому рекомендуется использование версии 3.4.1 или более поздней.

В дистрибутивах Linux Python 3 может быть уже установлен в вашей системе. Попробуйте ввести в консоли команду **python3**. Если команда не работает, попробуйте найти в репозитории и установить пакет python3.

Для языка Python существует много разных сред разработки, как коммерческих, так и бесплатных. Можно использовать IDLE – стандартную среду разработки для Python, но мы рекомендуем среду Wing IDE 101 – простую кроссплатформенную бесплатную среду для обучения. Сайт:
http://www.wingware.com/downloads/wingide-101

Последняя версия Wing IDE: 5.1.12.

Порядок установки важен: сначала установите интерпретатор, а только потом среду разработки, тогда она автоматически будет поддерживать установленный ранее интерпретатор. После установки Wing IDE из дистрибутива обязательно установите обновления к системе (меню “Help” – “Checkfor updates”).

§1. Структура программы. Понятие переменной. Оператор присваивания.

Для начинающих изучать программирование поясним, что программой называется последовательность команд (операторов), записанная на языке исполнителя программы. В нашем случае исполнителем программ будет Python-интерпретатор, поэтому программы нужно записывать согласно правилам синтаксиса данного языка.

В языке Python у программы нет никакой шаблонной структуры (обязательные к написанию вещи в начале и конце программы). Вам достаточно просто перечислить операторы в том, порядке, в котором вы хотите, чтобы они выполнились. Очень важное правило языка Python заключается в том, что каждый оператор нужно записывать в отдельной строке. Также большое значение имеет количество отступов от начала строки. Об отступах мы расскажем подробнее позднее.

Прежде чем начать рассмотрение операторов языка Python,познакомимся с очень важным в программировании понятием – понятием переменной. **Переменной** называется именованный объект определённого типа, который может менять своё значение в ходе выполнения программы. С каждой переменной связана область оперативной памяти компьютера, в которую сохраняется текущее значение переменной. Тип переменной определяет множество допустимых значений переменной, а также множество операций, которые можно применить к переменной. Рассмотрим примеры. Пусть переменная имеет целый числовой тип. Тогда

очевидно, что к ней можно применять различные арифметические действия, например, сложение, умножение или деление нацело. Если же переменная имеет, например, строковый тип, то понятно, что для неё будут определены совершенно другие операции (например, конкатенация или сцепление).

Для того чтобы задать переменной начальное значение или изменить её текущее значение используется **оператор присваивания**. Его синтаксис очень простой: записывается имя переменной, затем знак равенства « = », а затем значение, которое мы хотим присвоить данной переменной. Например, запись `a = 234` означает, что переменной `a` присвоено значение 234.

Переменной не обязательно присваивать константу (конкретное значение). Можно также копировать значение одной переменной в другую. Рассмотрим следующую последовательность операторов присваивания:

```
a = 234
v = a
```

В этом случае обе переменные (`a` и `v`) будут иметь значение 234.

Также можно присвоить переменной значение какого-то выражения (например, арифметического). В этом случае сначала всё вычисляется, а потом переменной присвоится результат вычислений выражения. Например:

```
a = 234 + 4
```

В этом примере переменной присвоится значение 238.

В состав выражения могут также входить переменные (записываются их имена). В этом случае, при вычислении подставляются их текущие значения. Например:

```
a = 234
v = a + 3
```

В этом примере переменной `v` присвоится значение 237, поскольку в момент вычисления текущее значение переменной `a` равно 234.

Ещё более интересный случай – использование самой переменной в выражении. Например:

```
a = 234
a = a + 3
v = a + 3
```

В этом примере происходит следующее. Сначала переменной `a` присваивается значение 234. Затем, при вычислении первого выражения используется именно это текущее значение. Результат вычислений 237, который и записывается в переменную `a`. Затем при вычислении второго выражения текущее значение переменной `a` уже равно 237, поэтому в переменную `v` запишется 240.

Теперь рассмотрим пример простейшей программы. Например, было задание вычислить длину гипотенузы прямоугольного треугольника по ее катетам. Запустите текстовый редактор среды программирования и напишите следующий текст:

```
a = 179
b = 197
c = (a ** 2 + b ** 2) ** 0.5
print(c)
```

В первой строке переменной `a` присваивается значение 179, затем переменной `b` присваивается значение 197, затем переменной `c` присваивается значение арифметического выражения, равного длине гипотенузы (как легко догадаться, ** – это операция возведения в степень, а возведение в степень 0,5 соответствует извлечению квадратного корня). После этого значение переменной `c` **выводится на экран**. Далее, найдите в среде кнопку «выполнить программу» (зеленый треугольник в wing). Если вы написали всё без ошибок, то на экране в отдельном окне появится результат работы программы, а в противном случае – сообщение об ошибке.

При выполнении программы значения вычисленных выражений не выводятся на экран, если программу специально не попросить их вывести. Для вывода данных на экран используется специальная функция `print`.

§2. Вывод данных. Функция print.

Стандартная функция `print` служит для вывода информации. Правило ее использования: после слова `print` в скобках через запятую перечисляются параметры, которые мы хотим вывести на экран (напечатать). Число этих параметров не ограничено. Запятая служит разделителем между параметрами:

```
print(параметр, параметр, ..., параметр)
```

Существует три вида параметров: *константы*, *переменные* и *выражения* (например, арифметические выражения).

Константы бывают числовые, логические и строковые. Числовые константы – это просто различные числа, целые и вещественные (вещественные числа ещё называют действительными). Любой текст, набранный с клавиатуры и заключенный в апострофы (одиночные кавычки) или двойные кавычки, называется строковой константой. Если в текст нам нужно поместить апостроф, например, в слове O'key, то его надо заключать в двойные кавычки и наоборот. Пока что ограничимся этими двумя типами констант, а о логических поговорим позднее.

Все параметры в операторе `print` независимы друг от друга, поэтому в одном и том же операторе могут встречаться параметры разных типов, в произвольном порядке.

При выполнении оператора вывода все параметры будут напечатаны в одной строке в том же порядке, в каком они перечислены в списке параметров. Любая константа числовая или строковая будет напечатана так, как вы ее написали в вызове `print` (в строковой константе начальный и конечный апострофы напечатаны не будут); вместо переменной на экране появится ее значение, а вместо арифметического выражения — результат его вычисления. Рассмотрим пример.

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

В данном случае будет напечатан текст

```
1 + 2 = 3
```

– сначала выводится значение переменной `a`, затем строка из знака “ + ”, затем значение переменной `b`, затем строка из знака “ = ”, наконец, значение суммы `a + b`.

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

```
print(a, b, c, sep = ':')
```

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

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

Для того, чтобы значение каждого параметра выводилось 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 слэша 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 слэша, 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 = '')
```

Здесь в апострофы помещена пустая строка (отсутствие символов), а не пробел!!!

§3. Ввод данных. Функция input.

Вернёмся к рассмотренному выше примеру простейшей программы (о прямоугольном треугольнике). Неудобство написанной программы заключается в том, что если мы захотим вычислить гипотенузу другого треугольника, то нам придётся менять значения катетов в тексте программы. Это неудобно, лучше, чтобы текст программы не менялся, а программа запрашивала бы у пользователя данные, необходимые для решения задачи, то есть запрашивала бы значения двух исходных переменных `a` и `b`. Для этого будем использовать функцию `input()`, которая считывает строку с клавиатуры и возвращает значение считанной строки, которое сразу же присвоим переменным:

```
a = input()
b = input()
```

Правда, функция `input` всегда возвращает текстовую строку, а нам нужно сделать так, чтобы переменные имели целочисленные значения. Поэтому сразу же после считывания выполним преобразование типов при помощи функции `int`, и запишем новые значения в переменные `a` и `b`.

```
a = int(a)
b = int(b)
```

Можно объединить считывание строк и преобразование типов, если вызывать функцию `int` для того значения, которое вернет функция `input`:

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

Теперь мы можем, не меняя исходного кода программы, многократно использовать ее для решения различных задач. Для этого нужно запустить программу () и после запуска программы ввести с клавиатуры два числа, нажимая после каждого числа клавишу Enter. Затем программа сама выведет результат.

§4. Основные типы данных. Операции. Преобразование типов.

Выше мы упоминали числовые и строковые типы переменных. Остановимся на них более подробно. Числа записываются последовательностью цифр, также перед числом может стоять знак минус, а строки записываются в одинарных кавычках. `2` и '2' – это разные объекты, первый объект – число, а второй – строка.

Кроме целых чисел есть и другой класс чисел: действительные (вещественные) числа, представляемые в виде десятичных дробей. Они записываются с использованием десятичной точки, например `2.0`. В каком-то смысле `2` и `2.0` имеют равные значение, но это – разные объекты.

Основные операции для чисел:

`A + B` — сумма;
`A - B` — разность;
`A * B` — произведение;
`A / B` — частное;
`A ** B` — возведение в степень. Полезно помнить, что квадратный корень из числа `x` — это `x ** 0.5`, а корень степени `n` – это `x ** (1 / n)`.

Есть также *унарный* вариант операции «–» (минус), то есть операция с одним аргументом. Она возвращает число, противоположное данному. Например: `-A`.

В выражении может встречаться много операций подряд. Как в этом случае определяется порядок действий? Например, чему будет равно `1 + 3 * 2 ** 3 + 1`? В данном

случае ответ будет 26, так как сначала выполняется возведение в степень, затем – умножение, затем — сложение.

Более общие правила определения приоритетов операций такие:

1. Выполняются возведения в степень справа налево, то есть `3 ** 3 ** 3` – это число 3^{27}, а не 27^3.
2. Выполняются унарные минусы (отрицания).
3. Выполняются умножения и деления слева направо. Операции умножения и деления имеют одинаковый приоритет.
4. Выполняются сложения и вычитания слева направо. Операции сложения и вычитания имеют одинаковый приоритет.

Основные операции над строками:

`A + B` — конкатенация (сцепка одной строки с другой, то есть строка `B` будет приписана в конец строки `A`);
`A * n` — повторение строки `A` `n` раз, значение `n` должно быть целого типа. Например, нельзя вычислить `'ABC' * 10.0`.
Примеры:

```
>>>  'Hello'+'world'
Helloworld
>>> A = 'ABC'
>>> A * 4
>>> ABCABCABCABC
```

Подумайте, что надо было сделать, чтобы слова Hello и world были разделены пробелом в результирующей строке.

Преобразование типов

Иногда бывает полезно целое число записать, как строку. И, наоборот, если строка состоит из цифр, то полезно эту строку представить в виде числа, чтобы дальше можно было выполнять с ней арифметические операции. Для этого используются функции, одноименные с именем типа, то есть, `int` для целых чисел, `float` для вещественных чисел, `str` для строк. Например, `int('123')` вернет целое число `123`, а `str(123)` вернет строку `'123'`.

Операции деления

Операция деления / всегда возвращает значение типа `float`, независимо от того, какого типа делимое и делитель (операнды). Также функция возведения в степень возвращает значение типа `float`, если показатель степени — отрицательное число. Но есть и специальная операция целочисленного деления, выполняющегося с отбрасыванием дробной части, которая обозначается //. Она возвращает целое число: целую часть частного. Обратите внимание на результат деления при отрицательном делимом. Например:

```
>>> 17 // 3
5
>>> -17 // 3
-6
```

Другая близкая ей операция: это операция взятия остатка от деления, обозначаемая `%`:

```
>>> 17 % 3
2
>>> -17 % 3
1
```

Обратите внимание, что 3 % 5 = 3, а не 0 или 2.

Пример полной программы

Пусть нам дана следующая задача: найти сумму цифр заданного четырёхзначного числа и вывести полученный результат. Решение этой задачи состоит из следующих этапов – ввод исходного числа, выделение из него отдельных цифр, нахождение суммы и вывод результата. Первое, третье и четвёртое из перечисленных действий являются очевидными. Рассмотрим, как делается второе (выделение цифр).

Очевидно, что для получения младшей цифры числа нужно взять остаток от деления его на 10, а для получения старшей цифры – нужно число нацело поделить на 1000. Для того же, чтобы получить вторую с конца цифру (число десятков), можно выполнить следующие два действия – сначала разделить число на 10 нацело, а потом из полученного числа взять остаток от деления на 10. Если применить описанные действия к числу 3254, то после деления нацело на 10 останется число 325, и остатком от деления на 10 будет цифра 5, которую нам как раз и требовалось найти. Теперь запишем полный код программы.

```
n = int(input())
c1 = n // 1000
c2 = n // 100 % 10
c3 = n // 10 % 10
c4 = n % 10
print('сумма цифр числа',n,'равна',c1 + c2 + c3 + c4)
```

<u>Задачи для решения находятся на сайте informatics.msk.ru</u>

Все задачи взяты из курса Д. П. Кириенко. Программирование на python. Целочисленная арифметика.

Зачетные задачи сгруппированы в разделе **Дистанционное обучение СУНЦ МГУ** (пароль выслан учащимся заочной школы). Раздел **Задачи на оператор присваивания и целочисленную арифметику**

Задачи можно сдавать и на других известных вам языках программирования, но мы рекомендуем хотя бы познакомиться с языком Python.

При решении этих задач можно использовать только операторы присваивания, рассмотренные арифметические операции, операторы ввода и вывода данных. Знакомым с условными операторами, операторами цикла и т.п. их использование **запрещено**.

Абсолютные номера задач для желающих (найти задачу по номеру можно с главной страницы сайта):

3457, 3458, 3460, 3465, 3468, 3470, 3473, 3474, 3475, 3477.

Тем, кто обучался в летней школе СУНЦ МГУ для 8классников, нужно решить те задачи из включенных в задание, которые еще не были решены ранее, при условии, что вы используете тот же самый логин.