**Изучить информацию и выполнить все задания**

**1.Анаконда, spyder:**

Пуск, Выполнить, spyder

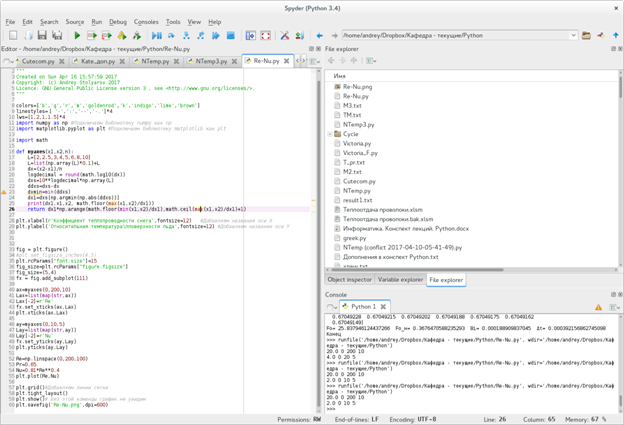
Spyder (Scientific PYthon Development EnviRonment) - научная интегрированная среда разработки на Python, в которую включено множество необходимых библиотек, и, что удобно, - со справочной системой. Для Spyder-а родным домом является Linux, но он адекватно работает и под Windows.

**Приёмы работы в Spyder IDE**

Немного подробнее рассмотрим Spyder IDE

Как и в большинстве IDE, запуск программы осуществляется с помощью клавиши https://konspekta.net/lektsiiorgimg/baza17/2120342604099.files/image001.png .

 Внешний вид Spyder IDE показан на рисунке 1.



Главное окно - это текстовый редактор. Он имеет синтаксическую подсветку (нельзя сказать, что самую богатую). Поскольку это полноценный текстовый редактор, то в нём можно при необходимости просматривать любые файлы, не только Python. Как и в большинстве текстовых редакторов, весьма важными для Python являются сочетания клавиш Tab и Shift+Tab, которые позволяют переместить выделенный блок текста на уровень влево/вправо. Ещё одно негативное замечание о редакторе: в текущих версиях Spyder под Linux иногда возникает проблема из-за несоответствия отступов, создаваемых знаком табуляции и пробелами. В итоге мы видим, что символы почти стоят друг над другом, но НЕ СОВСЕМ, что для Python означает ошибку Indentation Error. Чтобы убрать ошибку, приходится вручную удалять и заново делать отступы для «дефектного» блока.

Справа внизу находится окно консоли. Через консоли реализуется общение с машиной Pyhton, подробнее см. раздел «Ввод/вывод».

Вверху справа находятся вспомогательные окна, позволяющие просматривать списки и значения переменных (в режиме отладки), списки объектов (если они есть) и список файлов, достаточно удобно заменяющий при работе Файловый менеджер.

В оболочке Spyder есть неудобство, что отсутствует механизм ввода греческих букв, для решения этой проблемы можно использовать простую программу в разделе "Полезные советы".

При работе с текстом программы часто бывает удобно изменять размер шрифта. Это можно сделать с помощью комбинации Ctrl+ (Ctrl-) или в настройках редактора.

Отладка – отдельная область программирования и целое искусство. Её приёмы нужно перенимать, изучать, исследовать и изобретать.

Во-первых, отладка в любом языке основывается на понимании логики транслятора. У Турбо Паскаля, например, логика была настолько чёткая, что по коду ошибки было сразу понять, что нужно исправить. Про Python я так сказать не могу, тем не менее, нужно просто «привыкнуть к характеру» основных ошибок, например, что причину Syntax error часто нужно искать строчкой выше (незакрытая скобка или отсутствие знака «:» )

Во-вторых, использовать возможности отладчика (Debugger). Главная возможность - остановить выполнение программы и посмотреть значения переменных. Честно говоря, эргономика данной функции тоже не впечатлила: у меня обычно столько усилий уходит на выход из режима отладки, что пропадает желание и начинать.

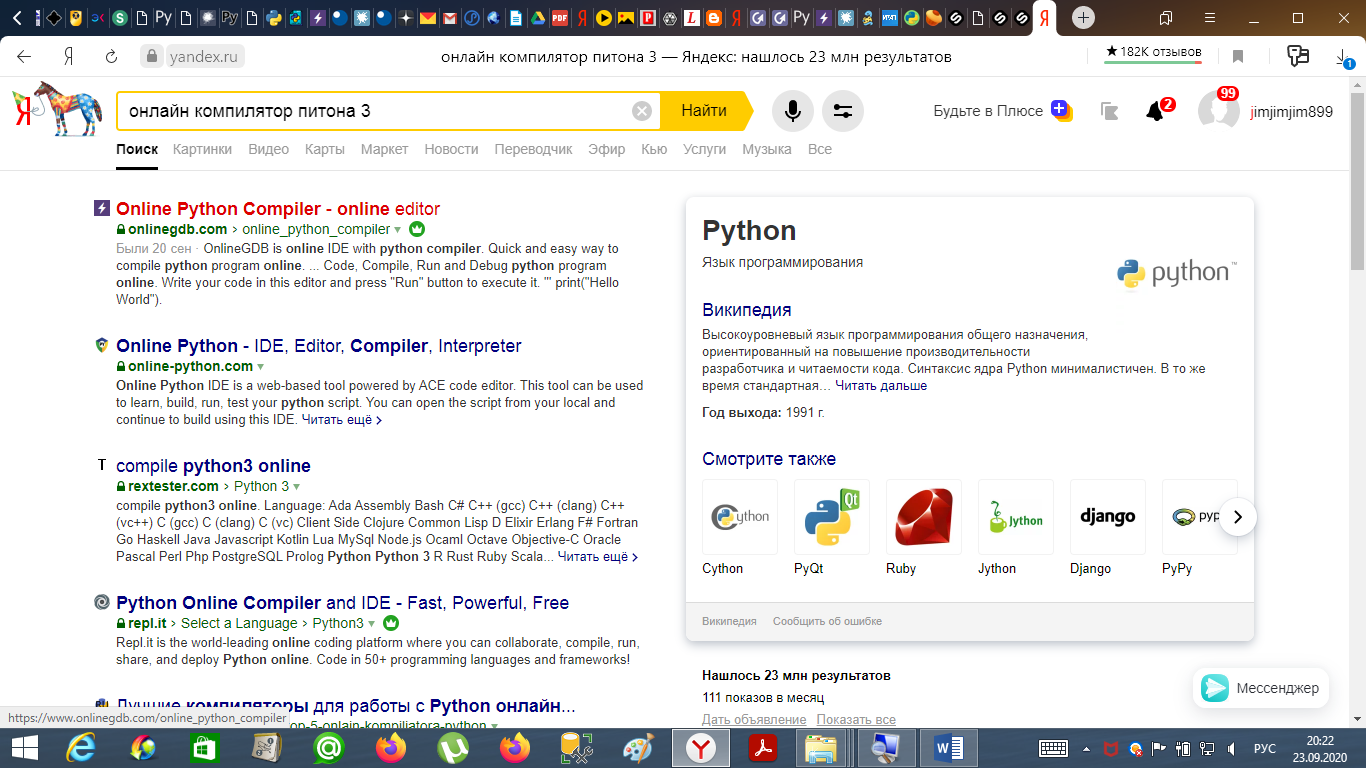
А вот ценным помощником является Менеджер переменных. Всячески рекомендую его использовать! Кстати, намекаю, что результаты иногда не обязательно даже выводить в консоль, можно так посмотреть в Менеджере. К сожалению, некоторые переменные в него не попадают. Щелчком правой кнопки мыши на Менеджере переменных вызывается меню, позволяющее редактировать переменные, строить графики и диаграммы и другие полезности.

В-третьих, Консоль! В консоли можно выполнять отдельные команды и смотреть результат в Менеджере переменных. Потом эти команды переносятся в программу.

**Справка в Spyder**

В Spyder хорошо работает система справки по комбинации Ctrl+I на любом объекте. Справка появляется в правом верхнем фрейме, там же, где вводятся Менеджеры переменных и файлов.

**Или Онлайн, например:**



**2. Ввод-вывод данных**

1. Вывод данных, команда print()
2. Ввод данных, команда input()
3. Решение задач

**Аннотация.** Ввод и вывод данных в языке Python. Несложные программы, которые умеют что-то выводить на экран (команда **print**) и считывать информацию с клавиатуры (команда input()).

## Вывод данных, команда print

Параметры print()

* **objects** – объект, который нужно вывести **\*** обозначает, что объектов может быть несколько;
* **sep** – разделяет объекты. Значение по умолчанию: **‘ ‘**;
* **end** – ставится после всех объектов;
* **file** – ожидается объект с методом **write** (***string***). Если значение не задано, для вывода объектов используется файл **sys.stdout**;
* **flush** – если задано значение **True**, поток принудительно сбрасывается в файл. Значение по умолчанию: **False**.

**Примечание: sep, end, file и flush** — это аргументы-ключевые слова. Если хотите воспользоваться аргументом **sep**, используйте:

Для вывода данных на экран используется команда print().

Внутри круглых скобок пишем, что хотим вывести на экран. Если это текст, то обязательно указываем его внутри кавычек. Кавычки могут быть одинарными или двойными. Только обязательно ставим одинаковые до и после текста.

Например, следующий код:

print('Мы изучаем язык Python')

выведет на экран текст:

Мы изучаем язык Python

**Запомни:** кавычки могут быть и одинарными, и двойными. Следующие две строки делают одно и тоже.

print('Python')

print("Python")

То, что мы пишем в круглых скобках у команды print(), называется **аргументами**или **параметрами**команды.

Команда print() позволяет указывать несколько аргументов, в таком случае их надо отделять запятыми.

Например, следующий код:

print('Скоро я', 'буду программировать', 'на языке', 'Python!')

выведет на экран текст:

Скоро я⎵буду программировать⎵на языке⎵Python!

Обратите внимание, в качестве разделителя при выводе данных между аргументами команды используется **символ пробел**. По умолчанию команда print() добавляет ровно один пробел между всеми своими аргументами. Например, следующая программа:

print('1', '2', '4', '8', '16')

выводит числа 1 2 4 8 16. Числа выведены ровно через один пробел.

**Запомни:** при написании кода, между аргументами команды print() после запятой мы ставим 1 символ пробел. Это общепринятое соглашение в языке Python. Этот символ пробела не влияет на вывод данных.

## Примечания

**Примечание 1.** Команда print() записывается только маленькими буквами, другое написание недопустимо, так как в Python строчные и заглавные буквы [различны](https://ru.wikipedia.org/wiki/%D0%A7%D1%83%D0%B2%D1%81%D1%82%D0%B2%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%BA_%D1%80%D0%B5%D0%B3%D0%B8%D1%81%D1%82%D1%80%D1%83_%D1%81%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB%D0%BE%D0%B2).

**Примечание 2.** Каждая последующая команда print() выводит указанный текст**с новой строки**. Например, следующая программа:

print('Какой хороший день!')

print('Работать мне не лень!')

выведет на экран две строки:

Какой хороший день!

Работать мне не лень!

**Примечание 3.** Команда print() с пустым списком аргументов просто вставляет новую пустую строку. Например:

print('Какой хороший день!')

print()

print('Работать мне не лень!')

выведет на экран три строки, одна из которых пустая:

Какой хороший день!

Работать мне не лень!

**Примечание 4.** Почему в Python можно использовать как одинарные, так и двойные кавычки для обрамления текста? Ответ очень прост — часто кавычки это часть текста. И чтобы Python мог правильно распознать такой текст, пользуемся разными:

* если в тексте нужны одинарные кавычки, то для обрамления такого текста используем двойные кавычки;
* если в тексте нужны двойные кавычки, то обрамляем его одинарными.

Результатом выполнения кода:

print('В тексте есть "двойные" кавычки')

print("В тексте есть 'одинарные' кавычки")

будет:

В тексте есть "двойные" кавычки

В тексте есть 'одинарные' кавычки

## Счастливая последовательность

Каждое число последовательности **4 8 15 16 23 42** напечатать на отдельной строке.

**Примечание.** Каждая последующая команда print() выводит указанный текст, начиная с новой строки.

## Звездный треугольник

Напишите программу, которая выводит указанный треугольник, состоящий из звездочек (\*).

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*

## Ввод данных, команда input

Все предыдущие программы выводили на экран текст, известный в момент написания программного кода. Однако программы могут работать с данными, которые станут известны только во время выполнения программы. Другими словами, программы могут считывать данные, а затем их использовать.

Для считывания данных в языке Python используется команда input().

Рассмотрим следующую программу:

print('Как тебя зовут?')

name = input()

print('Привет,', name)

Сначала программа распечатает текст на экран «Как тебя зовут?». Далее программа будет ждать от пользователя ввода данных. Ввод данных реализуется с помощью команды input().

Команда input()всегда пишется с круглыми скобками. Она работает так: когда программа доходит до места, где есть input(), она ждет, пока пользователь введёт текст с клавиатуры (ввод завершается нажатием клавиши Enter). Введенная строка подставляется на место input().

То есть, если вы ввели строку «Parrot», программа дальше будет работать так, как будто на месте input() было написано «Parrot».

Таким образом, input() получает от пользователя какие-то данные и вместо вызова подставляет **строковое значение**, в нашем случае записывает его в качестве значения **переменной**name.

**Пример.**

n = input("Введите имя. ")  
sn = input("Введите отчество. ")  
print("Здравствуйте, " + n + " " + sn + ".")

**Пример. Ввод значений через запятую**

# -\*- coding: utf-8 -\*-

num1, num2 = eval (input('Введите два числа '))

print(num1,num2)

**Пример. Ввод значений в одной строке через разделитель: пробел**

l1,l2 = [int(i) for i in input('Вводите числа через пробел: ').split()]

print(l1,l2)

# Вводите числа через пробел: 678 -87

#678 -87

**Пример. Ввод значений в одной строке через разделитель: запятую**

l1, l2 = input().split(', ')

l1, l2 = int(l1), int(l2)

print(l1, l2)

# 87, 67

# 87 67

**Пример. Вывод с использованием format**

print ('Программа, вычисляющая частное двух чисел. Введите два числа.')

a = int (input ())

b = int (input ())

print (a, '/', b, '=', '{:10.7f}'.format (a / b))

**Пример. Преобразование к типу float при вводе**

a = int(input("сколько дней в году? "))

print(a)

# в переменную a сохранится целое число

# сколько дней в году? 365

#365

a = float(input("введите первое число: "))

b = float(input("введите второе число: "))

c = a + b

print("сумма", a, "и", b, "равна", c)

#введите первое число: 15.5

#введите второе число: 12.3

#сумма 15.5 и 12.3 равна 27.8

Что такое переменные и что значит сохранить в переменной значение, подробнее поговорим чуть позже, а пока запоминаем:

Команда print() выводит на экран данные.

Команда input() считывает введенные с клавиатуры данные.

## Примечания

**Примечание.**Очень часто перед считыванием данных мы печатаем некоторый текст, чтобы пользователь, который вводит эти данные понимал, что именно от него требуется. Например, в программе

print('Как тебя зовут?')

name = input()

print('Привет,', name)

мы сначала выведем текст «Как тебя зовут?», а уже потом считаем данные.

Поскольку это достаточно распространённый сценарий, то в языке Python можно выводить текст, передавая его в качестве параметра в команду input(). Предыдущий код можно переписать так:

name = input('Как тебя зовут?')

print('Привет,', name)

## Приветствие

На вход программе подается строка текста – имя человека. Напишите программу, которая выводит на экран приветствие в виде слова «Привет» (без кавычек), после которого должна стоять запятая и пробел, а затем введенное имя.

**Формат входных данных**  
На вход программе подаётся одна строка — имя человека.

**Формат выходных данных**  
Программа должна вывести текст в соотвествии с условием задачи.

**Примечание.** Для считывания текста используйте команду input(), для печати текста на экране используйте команду print().

## Любимая команда

На вход программе подается строка текста – название футбольной команды. Напишите программу, которая повторяет ее на экране со словами « - чемпион!» (без кавычек).

**Формат входных данных**  
На вход программе подается название футбольной команды.

**Формат выходных данных**  
Программа должна вывести текст согласно условиям задачи.

**Примечание.** Для считывания текста используйте команду input(), для печати текста на экране используйте команду print().

## Повторяй за мной-1

Напишите программу, которая считывает три строки по очереди, а затем выводит их в той же последовательности, каждую на отдельной строчке.

**Формат входных данных**  
На вход программе подаются три строки, каждая на отдельной строке.

**Формат выходных данных**  
Программа должна вывести введенные строки в той же последовательности, каждую на отдельной строке.

**Примечание.** Для считывания текста используйте команду input(), для печати текста на экране используйте команду print().

# sep, end, переменные, PEP 8

1. Необязательные параметры команды print(): sep, end
2. Переменные
3. Стандарт PEP 8
4. Комментарии
5. Решение задач

**Аннотация.** Урок посвящен изучению необязательных параметров команды print() и более детальному разбору понятия переменных. Также мы поговорим о комментариях и стандарте PEP 8, которого придерживаются Python-программисты.

## Необязательные параметры команды print

По умолчанию команда print() принимает несколько аргументов (параметров), выводит их через один **пробел**, после чего **ставит перевод строки**. Это поведение можно изменить, используя необязательные именованные параметры sep (separator, разделитель) и end (окончание).

### Параметр sep

Рассмотрим следующий код:

print('a', 'b', 'c')

print('d', 'e', 'f')

Результатом выполнения такого кода будет:

a b c

d e f

Рассмотрим следующий код:

print('a', 'b', 'c', sep='\*')

print('d', 'e', 'f', sep='\*\*')

Результатом выполнения такого кода будет:

a\*b\*c

d\*\*e\*\*f

При первой печати в качестве строки разделителя между аргументами команды print() установлена строка sep='\*'.

При второй печати в качестве строки разделителя между аргументами команды print() установлена строка sep='\*\*'.

Таким образом, необязательный параметр sep команды print() позволяет установить строку, с помощью которой будут разделены аргументы при печати.

### Параметр end

Если перевод строки делать не нужно или требуется указать специальное окончание, то следует явно указать значение для параметра end.

Рассмотрим следующий код:

print('a', 'b', 'c', end='@')

print('d', 'e', 'f', end='@@')

Результатом выполнения такого кода будет:

a b c@d e f@@

По завершении первой печати вставлена строка @ вместо перевода строки. Аналогично, по завершении второй печати вставлена строка @@.

Параметры sep и end можно использовать вместе. Рассмотрим следующий код:

print('a', 'b', 'c', sep='\*', end='finish')

print('d', 'e', 'f', sep='\*\*', end='^\_\_^')

print('g', 'h', 'i', sep='+', end='%')

print('j', 'k', 'l', sep='-', end='#')

print('m', 'n', 'o', sep='/', end='!')

Результатом выполнения такого кода будет:

a\*b\*cfinishd\*\*e\*\*f^\_\_^g+h+i%j-k-l#m/n/o!

**Пример**

a = 5

print("a =", a, sep='00000', end='\n')

print("a =", a, sep='0', end=' ')

## Примечания

**Примечание 1.** Вызов команды print() с пустыми скобками ставит перевод строки.

**Примечание 2.** Последовательность символов \n называется управляющей последовательностью и задает перевод строки.

**Примечание 3.** Значения по умолчанию у параметров sep и end следующие:

sep=' ' # пробел

end='\n' # перевод строки

**Примечание 4.** Чтобы убрать все дополнительные выводимые символы, можно вызывать командуprint() так:

print('a', 'b', 'c', sep='', end='')

**Примечание 5.** Программный код

print('Python')

равнозначен коду

print('Python', end='\n')

**Примечание 6.** Если после вывода данных нужно более одного перевода строки, то необходимо использовать следующий код:

print('Python', end='\n\n\n')

Данная последовательность символов \n может применяться **в любой части строки, с целью переноса на другую строку аргументов, указанных за символом**\n .

Вариант примера:

print('Python', '\n', 'Rules', '\n', '4ever', sep='', end='!')

Результат будет следующим:  
Python

Rules

4ever!

Что будет выведено на экран в результате выполнения следующего кода?

print('31', '12', '2019', sep='-')

Ответ:

31-12-2019

Ответ:

Mercury\*Venus\*!Mars\*\*Jupiter?

Сколько строк будет распечатано в результате выполнения следующего кода?

print('a', 'b', 'c', sep='\*')

print('d', 'e', 'f', sep='\*\*', end='')

print('g', 'h', 'i', sep='+', end='%')

print('j', 'k', 'l', sep='-', end='\n')

print('m', 'n', 'o', sep='/', end='!')

print('p', 'q', 'r', sep='1', end='%')

print('s', 't', 'u', sep='&', end='\n')

print('v', 'w', 'x', sep='%')

print('y', 'z', sep='/', end='!')

**Примечание.** Последовательность символов \n задает перевод строки.

**Дополнительно**

**Пример – Проверка при вводе и форматный вывод чисел**

print('Введите значения H и L')

h=input()

l=input()

while type(h) and type(l)!=float:

try:

h=float(h)

l=float(l)

except ValueError:

print('Введите корректные значения H и L')

h=input()

l=input()

print ("Результат %e" %(h \* l))

print ("Результат %0.2f" %(h \* l))

print("Результат {:.0e}".format(h\*l))

print("Результат {:.1e}".format(h\*l))

print("Результат {:10.2f}".format(h\*l))

print("{:,.2f}".format(h\*l))

**Подстановки**

Еще один способ передачи форматируемых значений в строку представляет использование подстановок или специальных плейсхолдеров, на место которых вставляются определенные значения. Для форматирования мы можем использовать следующие плейсхолдеры:

* **s**: для вставки строк
* **d**: для вставки целых чисел
* **f**: для вставки дробных чисел. Для этого типа также можно определить через точку количество знаков в дробной части.
* **%**: умножает значение на 100 и добавляет знак процента
* **e**: выводит число в экспоненциальной записи

Общий синтаксис плейсхолдера следующий:

|  |  |
| --- | --- |
|  | {:плейсхолдер} |

В зависимости от плейсхолдера можно добавлять дополнительные параметры. Например, для форматирования чисел float можно использовать следующие параметры

|  |  |
| --- | --- |
|  | {:[количество\_символов][запятая][.число\_знаков\_в\_дробной\_части] плейсхолдер} |

При вызове метода format в него в качестве аргументов передаются значения, которые вставляются на место плейсхолдеров:

|  |  |
| --- | --- |
|  | welcome = "Hello {:s}"  name = "Tom"  formatted\_welcome = welcome.format(name)  print(formatted\_welcome)        # Hello Tom |

В качестве результата метод format() возвращает новую отформатированную строку.

Форматирование целых чисел:

|  |  |
| --- | --- |
|  | source = "{:d} символов"  number = 5  target = source.format(number)  print(target)   # 5 символов |

Если форматируемое число больше 999, то мы можем указать в определении плейсхолдера, что мы хотим использовать запятую в качестве разделителя разрядов:

|  |  |
| --- | --- |
|  | source = "{:,d} символов"  print(source.format(5000))   # 5,000 символов |

Для дробных чисел, то есть таких, которые представляют тип float, перед кодом плейсхолдера после точки можно указать, сколько знаков в дробной части мы хотим вывести:

|  |  |
| --- | --- |
|  | number = 23.8589578  print("{:.2f}".format(number))   # 23.86  print("{:.3f}".format(number))   # 23.859  print("{:.4f}".format(number))   # 23.8590  print("{:,.2f}".format(10001.23554))    # 10,001.24 |

Еще один параметр позволяет установить минимальную ширину форматируемого значения в символах:

|  |  |
| --- | --- |
|  | print("{:10.2f}".format(23.8589578))    #     23.86  print("{:8d}".format(25))               #      25 |

Для вывода процентов лучше воспользоваться кодом "%":

|  |  |
| --- | --- |
|  | number = .12345  print("{:%}".format(number))        # 12.345000%  print("{:.0%}".format(number))      # 12%  print("{:.1%}".format(number))      # 12.3% |

Для вывода числа в экспоненциальной записи применяется плейсхолдер "e":

|  |  |
| --- | --- |
|  | number = 12345.6789  print("{:e}".format(number))        # 1.234568e+04  print("{:.0e}".format(number))      # 1e+04  print("{:.1e}".format(number))      # 1.2e+04 |

**Форматирование без метода format**

В принципе для форматирования значений мы можем обойтись и без метода format, применяя следующий синтаксис:

|  |  |
| --- | --- |
| 1 | строка%(параметр1, параметр2,..параметрN) |

То есть в начале идет строка, которая содержит те же плейсхолдеры, которые были рассмотрены выше (за исключением плейсхолдера %), после строки ставится знак процента %, а затем список значений, которые вставляются в строку. Фактически знак процента представляют операцию, в результате которой образуется новая строка:

|  |  |
| --- | --- |
|  | info = "Имя: %s \t Возраст: %d" % ("Tom", 35)  print(info)   # Имя: Tom     Возраст: 35 |

Рядом с плейсхолдером указывается знак процента и в отличие от функции format здесь не требуются фигурные скобки.

Причем способы форматирования чисел здесь также применяются:

|  |  |
| --- | --- |
|  | number = 23.8589578  print("%0.2f  - %e" % (number, number))   # 23.86  - 2.385896e+01 |

## I like Python

Напишите программу, которая выводит на экран текст «I\*\*\*like\*\*\*Python» (без кавычек).

**Примечание.** Используйте необязательный параметр sep.

## Кастомный разделитель

Напишите программу, которая считывает строку-разделитель и три строки, а затем выводит указанные строки через разделитель.

**Формат входных данных**  
На вход программе подаётся строка-разделитель и три строки, каждая на отдельной строке.

**Формат выходных данных**  
Программа должна вывести введённые три строки через разделитель.

## Приветствие

Напишите программу, которая приветствует пользователя, выводя слово «Привет» (без кавычек), после которого должна стоять запятая и пробел, а затем введенное имя и **восклицательный знак**.

**Формат входных данных**  
На вход программе подаётся одна строка — имя пользователя.

**Формат выходных данных**  
Программа должна вывести текст в соотвествии с условием задачи.

**Примечание 1.** Перед восклицательным знаком не должно быть пробелов.

**Примечание 2.** Используйте необязательный параметр end.

## Переменные

В предыдущих упражнениях мы уже использовали **переменные**, но авторы курса до сих пор не объясняли, что же это такое и с чем его едят.

Мы знаем, что команда input() обозначает «подожди, пока пользователь введет что-нибудь с клавиатуры, и запомни то, что он ввёл». Просто так просить «запомнить» довольно бессмысленно: нам ведь потом надо будет как-то сказать компьютеру, чтобы он вспомнил то, что запомнил. Для этого используем **переменные**и пишем такой код:

variable\_name = input()

print('Вы ввели текст:', variable\_name)

Данный код означает: «Сохрани то, что ввёл пользователь, в памяти, и дальше это место в памяти мы будем называть именем variable\_name». Соответственно, команда print(variable\_name) означает: «Посмотри, что лежит в памяти, под именем variable\_name, и выведи это на экран».

Вот такие «места в памяти» называются переменными. Любая переменная имеет **имя и значение**.

### **Имя переменной**

1. В имени переменной используйте только латинские буквы a-z, A-Z, цифры и символ нижнего подчеркивания (\_);
2. Имя переменной не может начинаться с цифры;
3. Имя переменной по возможности должно отражать её назначение.

**Запомни:** Python — регистрочувствительный язык. Переменная name и Name — две совершенно разные переменные. Для именования переменных принято использовать стиль **lower\_case\_with\_underscores**(слова из маленьких букв с подчеркиваниями).

### **Значение переменной**

Значение переменной — сохраненная в ней информация. Это может быть текст, число и т.д.

Знак «=» это **оператор присваивания**. Он присваивает значение, которое находится справа от знака «равно», переменной, которая находится слева от знака «равно».

В нашем случае это то, что поместил в неё пользователь командой input(). Это текстовое значение — строка. То есть переменная сохраняет в себе строковое значение. Говорят, что переменная имеет строковый тип данных.

**Запомни:** интерпретатор ждет, что пользователь что-то введет с клавиатуры ровно столько раз, сколько команд input() встречается в программе. Каждый input() завершается нажатием **Enter** на клавиатуре.

Значения переменной, естественно, можно менять (переприсваивать).

print('Как тебя зовут?')

name = input()

print('Привет,', name)

name = 'Timur'

print('Привет,', name)

Оператор присваивания сообщает переменной то или иное значение независимо от того, была ли эта переменная введена раньше. Вы можете менять значение переменной, записав еще один оператор присваивания. Если у нас имеется переменная, мы можем делать с её значением все что угодно — например, присвоить другой переменной:

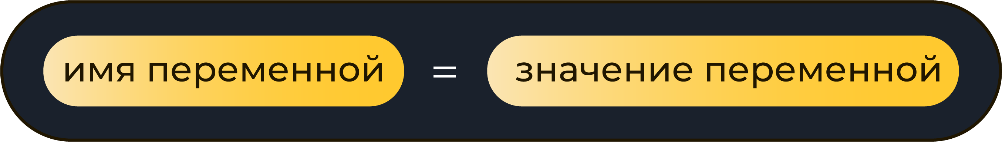
name1 = 'Тимур'

name2 = name1

print(name2)

Итак, если вы хотите, чтобы у вас была **переменная с каким-то именем и каким-то значением**, нужно написать на отдельной строчке:

<имя переменной> = <значение переменной>



Как только эта команда выполнится, в программе появится переменная с указанным значением.

**Запомни:** название переменной всегда должно быть **слева от знака равенства**.

Следующий код приведет к ошибке:  'Timur' = name.

## ****Множественное присваивание****

В языке Python можно за одну инструкцию присваивания изменять значение сразу нескольких переменных. Делается это так:

name, surname = 'Timur', 'Guev'

print('Имя:', name, 'Фамилия:', surname)

Этот код можно записать и так:

name = 'Timur'

surname = 'Guev'

print('Имя:', name, 'Фамилия:', surname)

Отличие двух способов состоит в том, что множественное присваивание в первом способе присваивает значение двум переменным одновременно.

Если требуется считать текст с клавиатуры и присвоить его в качестве значения переменным, то можно написать так:

name, surname = input(), input()

print('Имя:', name, 'Фамилия:', surname)

Если слева от знака «равно» в множественном присваивании должны стоять через запятую имена переменных, то справа могут стоять произвольные выражения, разделённые запятыми. Главное, чтобы слева и справа от знака присваивания было одинаковое число элементов.

Множественное присваивание удобно использовать, когда нужно обменять значения двух переменных. В Python это делается так:

name1 = 'Timur'

name2 = 'Gvido'

name1, name2 = name2, name1

**Пример.** **Позиционное присваивание**

x,y,z,w = 'abcd'

print(x)

#'a'

print(y)

#'b'

print(z)

#'c'

print(w)

#'d'

## Примечания

**Примечание 1.** Названия переменных ничего не говорят интерпретатору, и даже в очень хорошо названной переменной не появится нужное значение, если мы сами его туда не запишем.

**Примечание 2.** Новое значение переменной вытесняет старое. Важно представлять, чему равно значение переменной в каждый момент времени.

**Примечание 3.** Переменные можно вводить в любой момент (не только в самом начале программы).

**Примечание 4.** В качестве названия переменных запрещено использовать ключевые (зарезервированные) слова. К ключевым словам в языке Python относятся:

1. False;
2. True;
3. None;
4. and;
5. with;
6. as;
7. assert;
8. break;
9. class;
10. continue;
11. def;
12. del;
13. elif;
14. else;
15. except;
16. finally;
17. try;
18. for;
19. from;
20. global;
21. if;
22. import;
23. in;
24. is;
25. lambda;
26. nonlocal;
27. not;
28. or;
29. pass;
30. raise;
31. return;
32. while;
33. yield.

"Имя переменной по возможности должно отражать её назначение." Желательно акцентировать этот момент. Или программист использует осмысленные имена или в большой программе ему придётся писать много комментариев.

## PEP 8

При оформлении программ мы будем пользоваться **PEP 8 — Python Enhanced Proposal**. Этот документ предлагает единый и общепринятый стиль написания программ на языке Python. Документ создан по рекомендациям Гвидо Ван Россума, автора Python. Прочесть его на русском языке можно по [ссылке](https://pep8.ru/doc/pep8/).

## Некоторые рекомендации PEP 8

**Рекомендация 1.** Избегайте использования пробелов перед открывающей скобкой, после которой начинается список аргументов функции.

**Правильно:**

print('Follow PEP8!')

**Неправильно:**

print ('Follow PEP8!')

**Рекомендация 2.** После запятой нужен пробел.

**Правильно:**

print('PEP8', 'Rocks!')

**Неправильно:**

print('PEP8','Rocks!')

**Рекомендация 3.** Не отделяйте пробелами знак «равно», когда он употребляется для обозначения значения параметра по умолчанию.

**Правильно:**

print('My name', 'is', 'Python', sep='\*\*', end='+')

**Неправильно:**

print('My name', 'is', 'Python', sep = '\*\*', end = '+')

## Комментарии

До этого момента всё, что мы набирали в тексте наших программ, представляло собой команды для компьютера. Но в программу имеет смысл включать также примечания, описывающие, что она делает и как работает. Это может помочь вам или кому-то другому понять принцип работы программы спустя некоторое время.

Такие примечания называются **комментариями**.

### Однострочные комментарии

Любую строку можно превратить в комментарий, поместив перед ней символ #.

Рассмотрим следующий код:

# Это комментарий в программе на языке Python.

print('Python rocks!')

Если запустить эту программу, то получится:

Python rocks!

Первая строка при запуске игнорируется. Комментарий, начинающийся с символа #, предназначен только для автора программы и для тех, кто будет читать этот код.

### Комментарий в конце строки

Комментарий можно поместить после строки кода.

print('Python rocks!') # Это комментарий в программе на языке Python.

Комментарий начинается после символа #. Все, что находится до этого символа, представляет собой обычный код. Если запустить эту программу, получится:

Python rocks!

## Примечания

**Примечание 1.** Согласно стандарту PEP 8, комментарии должны отделяться хотя бы двумя пробелами от кода. Они должны начинаться с символа # и одного пробела.

**Правильно:**

# Далее будет напечатан текст

print('Comments in Python') # Печать текста с помощью команды print.

**Неправильно:**

#Далее будет напечатан текст

print('Comments in Python')#Печать текста с помощью команды print.

# Работа с целыми числами

1. Целочисленный тип данных
2. Преобразование строки к целому числу
3. Операции над целыми числами
4. Решение задач

**Аннотация.** Урок посвящен изучению работы с целыми числами. Изучим основные операции, а также научимся преобразовывать строки в числа.

## Целые числа

Все предыдущие программы, которые мы писали, работали с текстовыми данными. Действительно, команда input() считывает строку текста. Однако во многих случаях нам нужно работать именно с числами. Чтобы в Python создать переменную целого типа данных, нужно опустить кавычки при объявлении переменной. Рассмотрим следующий код:

num1 = 7 # num1 - это число

num2 = 10 # num2 - это число

num3 = num1 + num2 # num3 - это число

print(num3)

В результате выполнения такой программы будет выведено число 17.

**Запомни:** числа обозначаются без кавычек, а строки — с кавычками.

## Основные операции с числами

В языке Python, как и в математике, над числами можно совершать 4 основные операции (+, **−**, \*, /).

| **Оператор** | **Описание** |
| --- | --- |
| + | сложение |
| **−** | вычитание |
| \* | умножение |
| / | деление |

Рассмотрим следующую программу:

a = 3

b = 2

print(a + b)

print(a - b)

print(a \* b)

print(a / b)

Результатом выполнения такой программы будут числа:

5

1

6

1.5

## Порядок выполнения операций

В математике существует порядок выполнения операций, определяющий, какие операции должны выполняться раньше других, даже если в выражении они написаны правее. Порядок выполнения операций в Python аналогичен порядку выполнения операций, которые вы изучали на уроках математики.

Рассмотрим следующий код:

num1 = 2 + 3\*4

num2 = (2 + 3)\*4

print(num1)

print(num2)

Результатом выполнения такой программы будут числа 14 и 20. В переменной num1 будет храниться число 14, поскольку первым делом выполняется умножение, а уже затем сложение. В переменной num2 будет храниться число 20, поскольку скобки изменили приоритет выполнения операций.

**Запомни:** первым делом выполняется умножение или деление, затем сложение и вычитание. Для изменения порядка выполнения операций понадобятся скобки.

## Преобразование типов

**Преобразование строки к целому числу.** Для того, чтобы преобразовать строку к целому числу, мы используем команду int(). Рассмотрим следующий код:

s = '1992'

year = int(s)

Переменная s имеет строковый тип данных. С помощью команды int(), мы преобразовали строку к целому числу и записали результат в переменную с именем year.  **Мы используем новую команду (точнее, функцию) – int()**. Она означает следующее: «Возьми то, что указано в качестве аргумента в скобках, и преврати это в целое число». Таким образом, переменная year имеет целый тип данных, а переменная s – строковый тип данных.

Напишем программу, которая считывает два целых числа и выводит на экран их сумму. Следующий код тут не сработает:

num1 = input()

num2 = input()

print(num1 + num2)

Python считает, что в переменных num1 и num2 находится **текст**, поскольку команда input() по умолчанию считывает именно **текст**. Для того, чтобы явно указать, что требуется работать с переменными целого типа, надо написать так:

num1 = int(input())

num2 = int(input())

print(num1 + num2)

**Запомни:** для того, чтобы считать одно целое число, мы пишем следующий код:

num = int(input())

**Преобразование целого числа к строке.** Для того, чтобы преобразовать целое число в строку, мы используем команду str(). Рассмотрим следующий код:

num = 17

s = str(17)

Переменная num имеет целый тип данных. С помощью команды str() мы преобразовали целое число в строку и записали результат в переменную s. Таким образом, переменная num имеет целый тип данных, а переменная s – строковый тип данных.

## Примечания

**Примечание 1.** При работе с целочисленными переменными часто нужно осуществлять присвоение значению п

## Три последовательных числа

Напишите программу вывода на экран трех последовательно идущих чисел, каждое на отдельной строке. Первое число вводит пользователь, остальные числа вычисляются в программе.

**Формат входных данных**  
На вход программе подается одно целое число.

**Формат выходных данных**  
Программа должна вывести три последовательно идущих числа в соответствии с условием задачи.

## Сумма трёх чисел

Напишите программу, которая считывает три целых числа и выводит на экран их сумму. Каждое число записано в отдельной строке.

**Формат входных данных**  
На вход программе подаётся три целых числа, каждое на отдельной строке.

**Формат выходных данных**  
Программа должна вывести сумму введенных чисел.

a, b, c=int(input()), int(input()), int(input())

Или

print(int(input()) + int(input()) + int(input()))

или

print(int(input()) + int(input()) + int(input()))

или

print(sum([int(input()) for \_ in range(3)]))

или

sum = 0 for i in range(3): num = int(input()) sum+=num print(sum)

## Куб

Напишите программу, вычисляющую объём куба и площадь его полной поверхности, по введённому значению длины ребра.

**Формат входных данных**  
На вход программе подается одно целое число – длина ребра.

**Формат выходных данных**  
Программа должна вывести текст и числа в соответствии с условием задачи.

**Примечание.** Объём куба и площадь полной поверхности можно вычислить по формулам V = a^3, \, \, S = 6a^2*V*=*a*3,*S*=6*a*2.

**Input 1:**

25

**Output 1:**

Объем = 15625

Площадь полной поверхности = 3750

**Input 2:**

13

**Output 2:**

Объем = 2197

Площадь полной поверхности = 1014

**Input 3:**

56

**Output 3:**

Объем = 175616

Площадь полной поверхности = 18816

## Значение функции

Напишите программу вычисления значения функции f(a, \, b) = 3(a + b)^3 + 275b^2 - 127a - 41*f*(*a*,*b*) =3(*a*+*b*)3+275*b*2− 127*a*−41 по введеным целым значениям a*a* и b*b*.

**Формат входных данных**  
На вход программе подаётся два целых числа, каждое на отдельной строке. В первой строке — значение a*a*, во второй строке — значение b*b*.

**Формат выходных данных**  
Программа должна вывести значение функции по введённым числам a*a* и b*b*.

Приведите ваш вариант.

Например.

print((lambda a, b: 3\*(a+b)\*\*3+275\*b\*b-127\*a-41)(int(input()), int(input())))

или

print([3\*(a + b)\*\*3 + 275\*b\*\*2 - 127\*a - 41 for a, b in zip([int(input())], [int(input())])][0])

## Следующее и предыдущее

Напишите программу, которая считывает целое число, после чего на экран выводится следующее и предыдущее целое число с пояснительным текстом.

**Формат входных данных**  
На вход программе подаётся целое число.

**Формат выходных данных**  
Программа должна вывести текст согласно условию задачи.

**Input 1:**

20

**Output 1:**

Следующее за числом 20 число: 21

Для числа 20 предыдущее число: 19

**Input 2:**

0

**Output 2:**

Следующее за числом 0 число: 1

Для числа 0 предыдущее число: -1

Приведите свое решение

Например:

a = int(input())

print("Следующее за числом %d число: %d\nДля числа %d предыдущее число: %d" % (a,a+1,a,a-1))

## ****Стоимость покупки****

Напишите программу, которая считает стоимость **трех компьютеров**, состоящих из монитора, системного блока, клавиатуры и мыши.

**Формат входных данных**  
На вход программе подаётся четыре целых числа, каждое на отдельной строке. В первой строке — стоимость монитора, во второй строке — стоимость системного блока, в третье строке — стоимость клавиатуры и в четвертой строке — стоимость мыши.

**Формат выходных данных**  
Программа должна вывести одно число – стоимость покупки (трех компьютеров).

**Input 1:**

9900

55600

3999

2990

**Output 1:**

217467

**Input 2:**

15700

80550

12050

5890

**Output 2:**

342570

**Input 3:**

44990

123300

19600

8990

**Output 3:**

590640

Приведите свой вариант

Например:

print((sum((int(input()) for i in range(4))) \* 3))

или

n = [int(input()) for x in range(4)]

print(sum(n)\*3)

## Арифметическая прогрессия

**Арифметической прогрессией** называется последовательность чисел a\_1, a\_2, ..., a\_n*a*1​,*a*2​,...,*an*​, каждое из которых, начиная с a\_2*a*2​, получается из предыдущего прибавлением к нему одного и того же постоянного числа d*d* (разность прогрессии), то есть:

a\_n=a\_{n−1}+d*an*​=*an*−1​+*d*

Если известен первый член прогрессии и её разность, то n*n*-ый член арифметической прогрессии находится по формуле:

a\_n=a\_1+d(n-1)*an*​=*a*1​+*d*(*n*−1)

**Входные данные**  
На вход программе подаётся три целых числа: a\_1*a*1​, d*d* и n*n*, каждое на отдельной строке.

## Разделяй и властвуй

Напишите программу, которая считывает целое положительное число x*x* и выводит на экран последовательность чисел x, \, 2x, \, 3x,\,4x*x*,2*x*,3*x*,4*x* и 5x5*x*, разделённых тремя черточками.

**Формат входных данных**  
На вход программе подаётся целое положительное число.

**Формат выходных данных**  
Программа должна вывести текст согласно условию задачи.

**Input 1:**

7

**Output 1:**

7---14---21---28---35

**Input 2:**

1

**Output 2:**

1---2---3---4---5

**Input 3:**

5

**Output 3:**

5---10---15---20---25

**Выходные данные**  
Программа должна вывести n*n*-ый член арифметической прогрессии.

**Input 1:**

1

1

10

**Output 1:**

10

**Input 2:**

-1

1

2

**Output 2:**

0

**Input 3:**

100

50

1

**Output 3:**

100

# работа с целыми числами

1. Операция возведения в степень
2. Операция нахождения остатка
3. Операция целочисленного деления
4. Обработка цифр числа
5. Решение задач

**Аннотация.** Урок посвящен дополнительным операциям при работе с целыми числами. Изучим дополнительные операции, а также научимся обрабатывать цифры целого числа.

## Дополнительные операции

Мы познакомились с 4 основными математическими операциями в языке Python: сумма (+)(+), разность (-)(−), произведение (\*)(∗) и частное (/)(/). Добавив ещё 33 операции, мы получим инструментарий, достаточный для написания 99\%99% программ.

|  |  |
| --- | --- |
| **Оператор** | **Описание** |
| \*\* | Возведение в степень |
| % | Остаток от деления |
| // | Целочисленное деление |

### Возведение в степень

Оператор возведения в степень a \*\* n возводит число a*a* в степень n*n*. Рассмотрим работу данного оператора на примерах:

print(2 \*\* 0)

print(2 \*\* 1)

print(2 \*\* 2)

print(2 \*\* 3)

print(2 \*\* (-1))

Результатом выполнения такой программы будет:

1

2

4

8

0.5

**Обратите внимание:** оператор возведения в степень \*\* может возводить не только в положительную степень, но и в отрицательную. Аналогично, основание степени a*a* также может быть числом отрицательным.

**Запомни:** в отличии от многих языков программирования, в языке Python для возведения в степень используется оператор \*\*, а не ^.

### Целочисленное деление

Для положительных чисел оператор целочисленного деления ведёт себя как обычное деление, за исключением того, что он отбрасывает десятичную часть результата. Рассмотрим работу данного оператора на примерах:

print(10 // 3)

print(10 // 4)

print(10 // 5)

print(10 // 6)

print(10 // 12)

Результатом выполнения такой программы будет:

3

2

2

1

0

При делении отрицательных чисел необходимо помнить, что результат целочисленного деления не превосходит частное. Другими словами, округление берётся в меньшую сторону (число -4−4 меньше чем число -3−3).

Результатом работы следующей программы:

print(10 // 3)

print(-10 // 3)

будут числа:

3 # округление в меньшую сторону

-4 # округление в меньшую сторону

### Деление с остатком

Оператор деления с остатком возвращает остаток от деления двух целых чисел. Рассмотрим работу данного оператора на примерах:

print(10 % 3)

print(10 % 4)

print(10 % 5)

print(10 % 6)

print(10 % 12)

Результатом выполнения такой программы будет:

1

2

0

4

10

**Запомни:** при нахождении остатка от деления на целое число n*n* может получаться результат 0, \, 1, \, 2, \, \ldots n -10,1,2,…*n* −1. Например, при делении на 22 возможны остатки 0, \, 10,1, при делении на 33 возможны остатки 0, \, 1, \, 20,1,2 и т.д.

## Примечания

**Примечание 1.** Оператор нахождения остатка очень полезен при решении многих задач. Например, число делится на n*n* нацело тогда и только тогда, когда остаток от деления на n*n* равен 00.

**Примечание 2.** Операторы // и % имеют такой же приоритет, как и операторы умножения и обычного деления.

**Примечание 3.** Наивысший приоритет имеет оператор возведения в степень \*\*.

**Примечание 4.** Полезно прочитать про [модулярную арифметику](https://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E) в математике.