Программирование на языке Python.  
Продвинутый уровень

**Модуль 1. Основные конструкции языка Python**

*Тема 1.1. Основы языка Python*

***Лекционное занятие (1 час)***

*Материал для учителя*

Язык программирования (ЯП) - формальный метод записи компьютерных программ.

Первым программируемым устройством считают разностную машину Чарльза Бэббиджа, описанную в 1835 г. С того времени создано более 8000 языков программирования.

В 40-50 г.г. ХХ в. появляются машинно-зависимые языки программирования, то есть для каждого вида ЭВМ используется свой языке. Например, Планкалкюль, описанный К.Цузе в 1948 г. был разработан специально для ЭВМ Z4.

В начале 50-х г.г. ХХ в. появляется язык Ассемблер, который представляет собой низкоуровневый язык программирования, где каждая команда преобразовывалась в машинный код с помощью специальной программы - компилятора.

С середины 50-ых гг. XX в. начали создавать первые языки программирования высокого уровня, которые были машинно-независимыми, то есть не привязанными к определенному типу ЭВМ. Одним из первых - FORTRAN (FORmula TRANslator), который создан в период с 1954 по 1957г. в IBM под руководством Джона Бэкуса для технических и научных задач. Fortran стал первым относительно массовым языком программирования: к 1963г. существовало уже 40 компиляторов для различных машин. В 1958г. совместными усилиями американских и европейских ученых был разработан язык ALGOL, ставший основой многих современных языков программирования, таких как Pascal, C, C ++ и Java. В 1958 г. появляется язык Lisp, а в 1959 г. – язык Cobol.

Язык Basic (Beginner’s All-purpose Symbolic Instruction Code) создан профессорами Дартмутского колледжа Т. Куртцем и Дж. Кемени в 1965 году для обучения студентов, незнакомых с вычислительной техникой.

В 1970 г. профессор Никлаус Вирт создал язык Pascal.

В 1972 г. Д. Ритчи был разработан язык С. Предшественником его считают язык Би, разработанный в 1969 в AT&T Bell Telephone Laboratories. С разрабатывался в Bell Labs в период с 1969 по 1973 годы. Цель создания языка - написание системного ПО, в частности операционных систем. На нем была написана, например, ОС UNIX. Ввиду этого в язык формировался удобным для управления аппаратными компонентами компьютера.

В 1978 г. Б.Керниган и Д. Ритчи опубликовали первую редакцию книги «Язык программирования Си», которая содержала следующие особенности языка: структуры, длинное целое, целое без знака, оператор +=.

В 1983 г. Б. Страуструп модифицировал язык C в Bell Labs, C ++ - это расширение C с такими улучшениями, как классы, виртуальные функции и шаблоны.

Язык программирования Python разработан в 1990 г. Гвидо ван Россумом на базе языка АВС, созданного в центре математики и информатики в Нидерландах. Изначально в языке не была реализована ООП. Затем в феврале 1991 года был опубликован исходный код языка. В нем уже заложены принципы ООП. Версия Python 2.0 выпущена в 2000г., а в 2008г. - версия Python 3.0, которая не полностью совместима со второй, а потому эти ветви стали развиваться параллельно. Сейчас разработчики предлагают уже версию 3.11. Синтаксис языка достаточно интересен. Язык не обладает избыточностью, код короток, нет нужды использовать «;», если конец строки, а вложенность выделяют отступами.

Можно отметить следующие особенности языка:

• динамическая типизация, то есть не обязательно указывать тип переменной. Операнды разных типов, участвующие в одной операции, автоматически приводится к нужному по определённым правилам. Тип данных привязан к значению, а не к переменной.

• Функции в языке могут возвращать список значений.

• Язык относится к группе интерпретируемых, написанный код не требует компиляции.

• Язык реализует несколько парадигм программирования, сфера его применения достаточно велика: Системные утилиты, Web-сайты приложения для научных расчетов, Прототипы, Игры, Мобильные приложения (kivy).

Минусы языка - низкое быстродействие и недостаточность средств анализа кода.

В этом курсе предлагается работа с Python 3.x. Рассмотрим установку данного пакета в операционных системах семейства Windows.

Для установки нужно в браузере перейти на официальную страницу http://www.python.org/download/ и загрузите последнюю версию Python 3. Установка производится стандартным способом так же, как и для любых других программ Windows.

Для того чтобы работать с Python из командной строки Windows, необходимо установить переменную PATH. Это можно сделать при установки выбрав соответствующий пункт Add Python 3.x to PATH. (Рисунок 1).

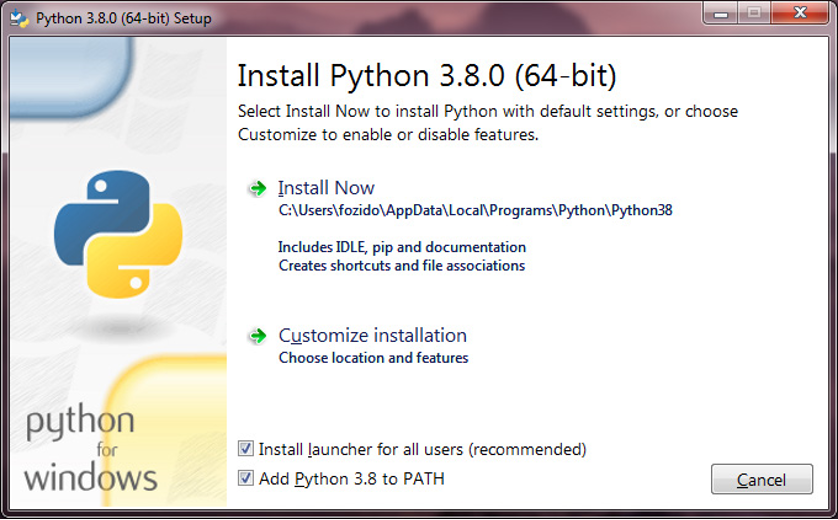


Рисунок 1 – Окно установки Python 3.x для Windows.

Для ее настройки необходимо перейти в свойство компьютера или нажать кнопку «Пуск» далее выбрать «Панель Управления» далее «Система и безопасность» далее «Система». Потом нажать кнопку «Дополнительные параметры системы» слева, а затем выбрать вкладку «Дополнительно». Внизу нажать кнопку «Переменные среды» и в отделе «Системные переменные» найти PATH, выбрать её и нажать «Редактировать».

Далее перейти к концу строки в поле «Значение переменной» и дописать: C:\Python3x. Где x – это номер версии, например: ;C:\Python38.

Если значение переменной было %SystemRoot%\system32;, теперь оно примет вид %SystemRoot%\system32;C:\Python3x. Если все шаги установки выполнены правильно, то можно запускать интерпретатор из командной строки в Windows. Чтобы открыть терминал в Windows, нажмите кнопку «Пуск» и выберите «Выполнить». В появившемся диалоговом окне наберите cmd и нажмите Enter. Далее набираем слово python и проверяем, нет ли ошибок. (Рисунок 2).

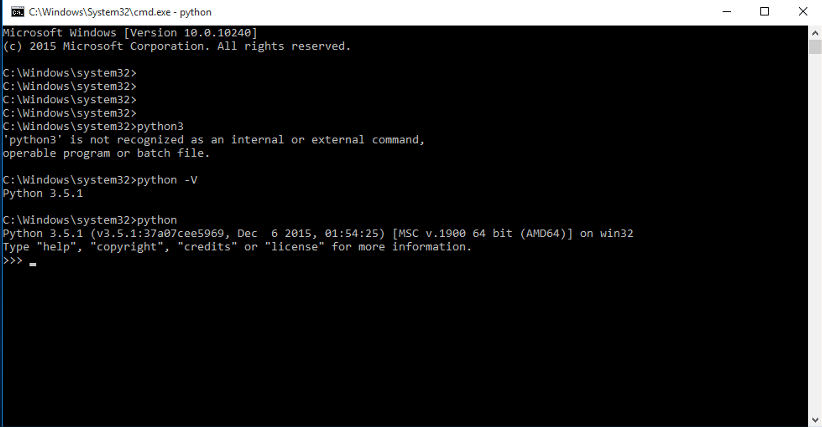


Рисунок 2 – Работа Python 3.x в командной строке Windows

Для пользователей Windows чтобы запустить интерпретатор в командной строке, нужно правильно установить переменную PATH. Для открытия командной строки в Windows, нужно перейти в меню «Пуск» и нажмите «Выполнить…». В появившемся диалоговом окне ввести «cmd» и нажать клавишу Enter; теперь у вас будет всё необходимое для начала работы с python3 в командной строке.

Альтернативным способом можно использовать IDLE которая устанавливается вместе с языком Python, необходимо перейти «Пуск» далее «Программы» далее «Python 3.x» и выбрать «IDLE (Python GUI)» (рисунок 3).

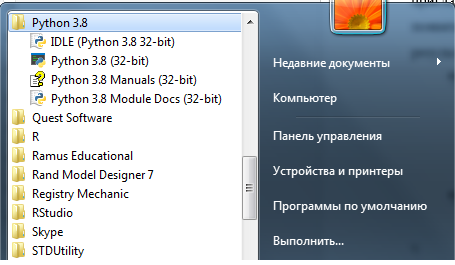


Рисунок 3 – Запуск Python 3.x

Как только будет запущен python3, на экране будет символ >>> в начале строки, где вы можете что-то набирать. Это и называется командной строкой интерпретатора Python 3.x (Рисунок – 4). Попробуем напечатать в окне приглашения print('Привет') и нажать клавишу Enter. В результате должны появиться слова «Привет». Как видим синтаксис языка подсвечивается, результат работы программы выделяется синим цветом.

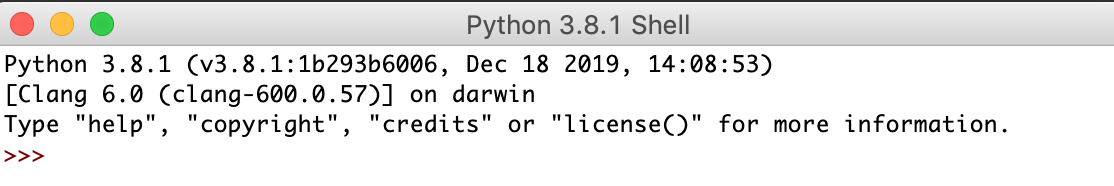


Рисунок 6 – Окно IDLE (Python GUI)



Рисунок 4 – Окно IDLE (Python GUI)

Стоит также заметить, что Python 3.x выдаёт результат работы строки немедленно. Но не всегда удобно с точки зрения работы с разными программами набирать их в командной строке интерпретатора. Очень важной составляющей является сохранения программы в файл, чтобы потом можно было запускать её неограниченное количество раз.

На практике в современном программирование используются различные редакторы кода. Для удобного написания программ на Python 3, нам понадобиться редактор кода для работы с файлами наших программ. Стоит очень ответственно подойти к выбору редактора кода. Тут нужно отметить, что хороший редактор поможет вам легко писать программы на Python 3, сделает это понятно, быстро, комфортно.

Цель работы: Ознакомление со средой программирования Python и запуском первых программ.

***Практикум (1 час)***

    1. Проверить, установлен ли на компьютере/ ноутбуке IDE PyCharm / IDE Python 3.x.

    2. Ознакомится с дидактическим материалом, посвященным установки языка Python,  установке и первичной настройке IDE PyCharm.

    3. Создайте первую программу на языке Python.

    Хорошей традицией при изучении языка программирования является написание первой вашей программы с простым приветствием на экране «Привет, я изучаю Python!».

    На жёстком диске создадите каталог с названием My\_projects. Запустить программу PyCharm/ IDE Python 3.x. Выбрать вкладку Открыть и указать вашу папку  My\_projects. Далее на экране перед пользователем открывается рабочее окно нашего проекта (Рисунок 1).

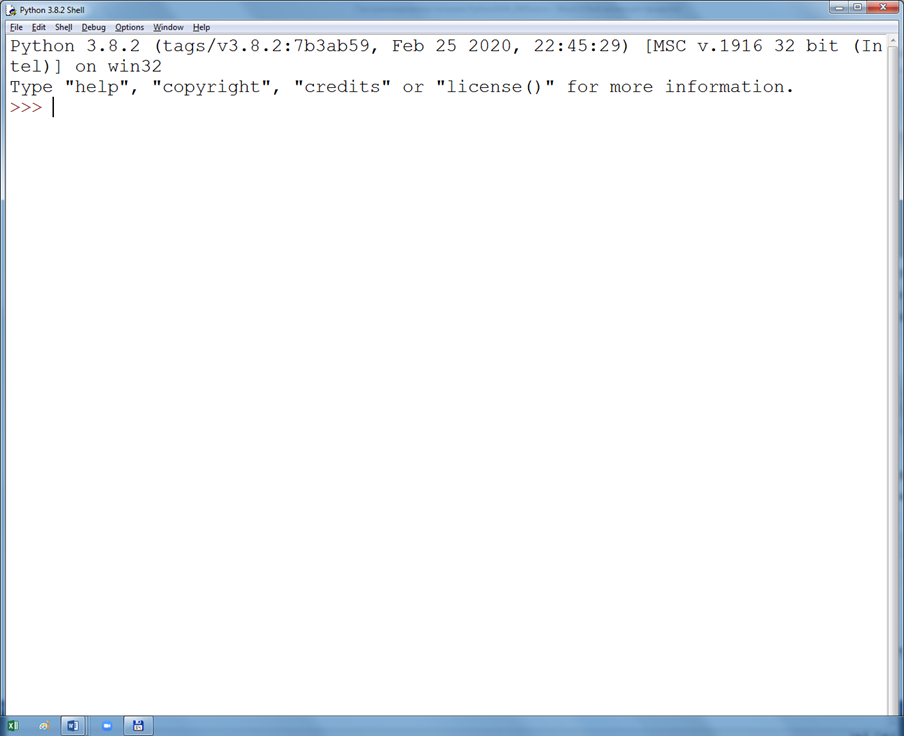


Рисунок 1 – Вид программы

Выберите команду File→New file.

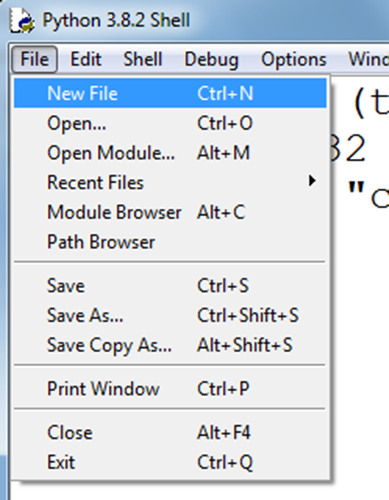


Рисунок 2 – Выбор команды

    В скобках в кавычках - текст, который нужно вывести в результате работы нашей программы (рис.3).

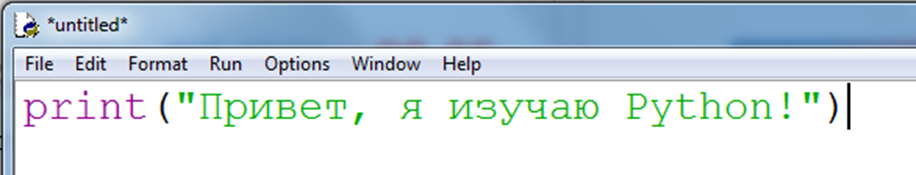
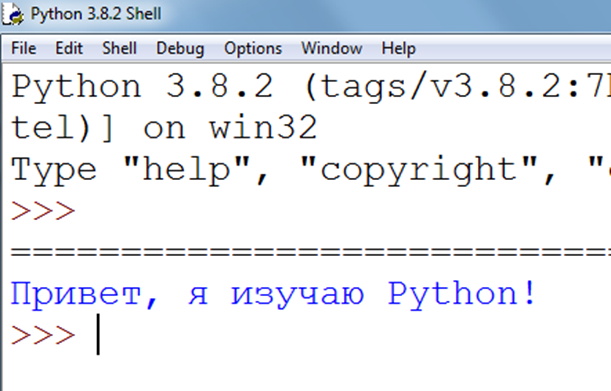


Рисунок 3 – Ввод программы

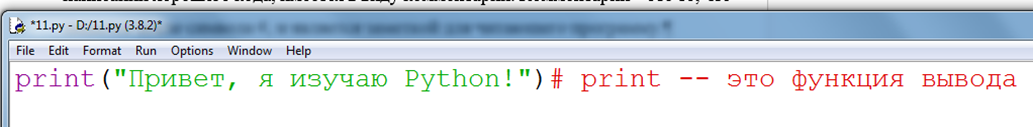
    После такого как выполнен набор кода программы, необходимо ее запустить и получить результат. Для запуска выберите команду Run→Run Module (F5). Если ваша программа не сохранена, то IDE предложит сохранить ее. Выполите сохранение в ранее созданную папку.

    Результаты работы программы представлен ниже (рис. 4).

Рисунок 4 – Результат работы программы

    Далее предлагается рассмотреть следующий механизм, помогающий в написании хорошего кода, имеется в виду комментарии. Комментарии – это то, что пишется после символа #, и является заметкой для читающего программу.

    print(Привет, я изучаю Python!) # print -- это функция вывода

Рисунок 5 – Текст программы

    Далее предлагается рассмотреть такой случай когда в строку вывода требуется добавить какие-либо данные. Для этого в Python применяют метод format().

    age=10

    print("Привет, я изучаю Python {0} лет!".format(age))

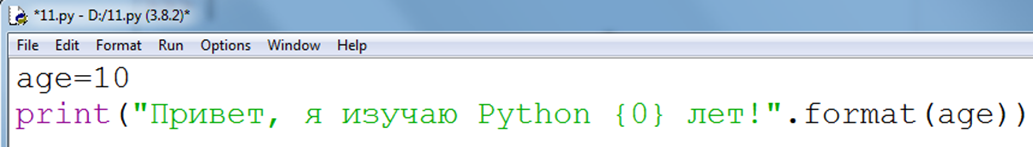


Рисунок 6 – Текст программы

    Далее запускаем программу на выполнение (рис.7).

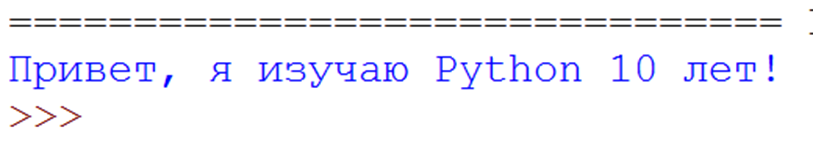


Рисунок 7 – Результат работы программы

Самостоятельно выполните следующие задания:

1. Создайте новый файл. В нем запишите текст программы, которая выводит на экран текст: Python – один из современных языков программирования.

***Тема 1.2. Ввод и вывод данных***

***Лекционное занятие (1 час)***

Любую программу языке Python можно представить, как набор лексем (допустимых символов), записанных в определенном порядке и по определенным правилам. Лексема может представлять собой: комментарии; литералы, знаки пунктуации, переменные, специальные ключевые слова.

Программа на языке Python может содержать достаточное количество комментариев, каждый комментарий начинается с символа решетка «#».

Литералы представляют собой значения, заданные в коде программы, например, числа (25) или строки (“привет”). В языке Python представлены некоторые встроенные типы объектов. В языке Python используется динамическая типизация (типы данных определяется автоматически, и их не требуется объявлять в программном коде), но при этом он является языком со строгой типизацией (вы сможете выполнять над объектом только те операции, которые применимы к его типу).

Если говорить об использовании знаков пунктуации, то стоит отметить, что каждая строка в программе на языке Python не должна заканчиваться точкой запитой, как, например, в С++, но если есть необходимость записать несколько инструкций в одну строчку, то их можно разделять точкой с запятой.

При написании программ в языке Python придерживаются так называемого PEP8 – стиль кода в языке Python. Опишем некоторые его основные положения. С полным положением можно ознакомится по ссылке <https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html>.

* Используйте 4 пробела на один уровень отступа. В старом коде, который вы не хотите трогать, можно продолжить пользоваться 8 пробелами для отступа.
* Никогда не смешивайте символы табуляции и пробелы.
* Ограничьте максимальную длину строки 79 символами.
* Отделяйте функции (верхнего уровня, не функции внутри функций) и определения классов двумя пустыми строчками.
* Используйте (без энтузиазма) пустые строки в коде функций, чтобы отделить друг от друга логические части.

Переменная или идентификатор используются для хранения данных различного типа. В языке Python нет специального раздела описания переменных, в котором указывается тип переменной перед ее первым использованием. Есть определенные правила для задания имен переменных языке Python: это последовательно букв, которая не может начинаться с цифры, но может содержать символ подчеркивания (\_). Имена переменных чувствительны к регистру. Имена переменных не могут совпадать со специальными ключевыми словами.

Ключевые слова в языке Python имеют специальное назначение и представляют собой управляющие конструкции языка. Например, and, break, for и т.д.

При составлении программ лексемы объединяются в синтаксические конструкции, которые могут вкладываться друг в друга. В результате могут образовываться блочные конструкции, каждый блок кода начинается двоеточием (:), а тело блока выделяется обязательным отступом в виде 4-х пробелов. Обычно среда программирования сразу делает отступ для блока после двоеточия. В зависимости от используемой среды программирования, блоки могут иметь визуальное выделение.

Часто при решении задач необходимо выполнять ввод данных с клавиатуры и вывод данных на экран. Рассмотрим работу соответствующих операторов в языке Python.

Ввод с клавиатуры осуществляется с помощью оператора print. Формат оператора:

print()

У команды следующие аргументы: **end** и **sep**.

**end** – задает действие по завершении print (по умолчанию end=”**\n”** – новая строка). Можно задать и свое действие: **end**='<разделитель> '.

x=2; y=12; z=-5

print(x,y,z)

Результаты работы программы представлены на рисунке 1:

|  |  |
| --- | --- |
|  | Рисунок 1 – Результат работы программы |

x=2; y=12; z=-5

print(z)

print(x,y)

Результаты работы программы представлены ниже на рисунке 2:

|  |  |
| --- | --- |
|  | Рисунок 2 – Результат работы программы |

x=2; y=12; z=-5

print(x)

print("z=", z, end='')

print(y)

Результаты работы программы представлены ниже на рисунок 3:

|  |  |
| --- | --- |
|  | Рисунок 3 – Результат работы программы |

Выводить на экран можно не только значения переменных, но и значения выражений.

x=3; y=6; z=-10

print(x)

x+=6

print("x=",x,end='')

print(x\*z)

Результаты работы программы представлены ниже на рисунке 4:

|  |  |
| --- | --- |
|  | Рисунок 4 – Результат работы программы |

Или еще изменим программу:

x=3; y=6; z=-10

print(x)

x+=6

print("x=",x)

print("x\*z=",x\*z)

Результат работы – на рисунке 5.

|  |  |
| --- | --- |
|  | Рисунок 5 – Результат работы программы |

Еще один параметр команды print – sep. sep – это может быть строка, которую необходимо вставлять между значениями, по умолчанию представлена пробелом.

x=9; y=11

print(x,y,sep='+')

Результат работы представлен ниже (рис.6).

|  |  |
| --- | --- |
|  | Рисунок 6 – Результат работы программы |

Значение **\n** перенесет каждое слово на новую строку.

Также в языке Python можно использовать форматированный вывод. Для этого необходимо использовать встроенную функцию format(). Ее синтаксис:

<строка>.format( <формат>)

«строка» представляет собой значение для форматированного вывода, «формат» – спецификации формата 'Mini-Language'.

a = 7; b = 22; c=a+b

print( "{0}+{1}={2}".format(a, b, c) )

Результаты работы программы представлены ниже на рисунке 6:

|  |  |
| --- | --- |
|  | Рисунок 7 – Результат работы программы |

Для ввода данных с клавиатуры используется встроенная функция input(), возвращающая в качестве результата строку! Покажем это на следующим примере.

a=input()

c=input()

b=a+c

print(b)

Результат вывода на экран показан на следующем рисунке:

|  |  |
| --- | --- |
|  | Рисунок 8 – Результат работы программы |

То есть вывелась не сумма числе 6 и 7, а сумма (конкатенация) строк «6» и «7».

Если нужно ввести числа, то следует использовать встроенные функции int и float.

Функция int преобразует строку в целое число:

a=int(input())

c=int(input())

b=a+c

print(b)

|  |  |
| --- | --- |
|  | Рисунок 9 – Результат работы программы |

У функции input можно указать аргумент- «приглашение»:

a=int(input("a="))

c=int(input("c="))

b=a+c

print(b)

|  |  |
| --- | --- |
|  | Рисунок 10 – Результат работы программы |

**Практикум 2.1 (1 час)**

  Цель работы: Ознакомление с основами написания программ на языке  программирования Python, работа с операторами ввода/вывода данных.

    1. Откройте среду разработки Python.

    2. Составить программу ввода с клавиатуры трех целых чисел. Выведите на экран их через пробел в одну строку.

    3. Составить программу ввода с клавиатуры трех целых чисел. Выведите на экран их в столбик, одно под другим.

    4. Составить программу ввода с клавиатуры двух вещественных чисел. Выведите на экран их сумму.

**Практикум 2.2 (2 часа)**

*Материалы для ученика*

    Цель работы: Ознакомление с основами написания программ на языке программирования Python, работа с операторами ввода/вывода данных.

    1. Откройте среду разработки Python.

    2. Составить программу ввода с клавиатуры двух целых чисел, так чтобы пользователь понимал, ввод какой переменный осуществляет. Выведите на экран значение их суммы, разности и произведения в следующем виде:

    5+6=11 5-6=-1 5\*6=30.

    3. Составить программу нахождения периметра и площади прямоугольника (значения длин сторон вводятся с клавиатуры).

    4. Составьте программу вычисления площади прямоугольного треугольника, заданного своими катетами.

*Тема 1.3. Типы данных*

***Лекционное занятие (2 часа)***

*Материал для учителя*

В этом же разделе упомянем работу со строенными математическими функциями. Все математические функции в языке Python объединены модуль math. Данный модель при необходимости нужно импортировать с помощью команды import:

import math

После подключения можно использовать множество математических функция Python.

В составе math присутствуют несколько известных математических констант:

Таблица 2 – Математические константы языка Python

|  |  |
| --- | --- |
| **Обозначение** | **Описание** |
| math.pi | Математическая константа π = 3.141592…. |
| math.e | Число Эйлера е=2,71828… |
| math.inf | Положительная бесконечность |

Опишем наиболее часто используемые функции, входящие в состав модуля math.

Таблица 3 – Математические функции языка Python

|  |  |
| --- | --- |
| **Обозначение** | **Описание** |
| math.exp(x) | Экспонента числа ех |
| math.log(X, [А]) | Логарифм X по основанию А. Если А не указан, вычисляется натуральный логарифм |
| math.pow(X, n) | Возведение в степень Хn |
| math.sqrt(x) | Квадратный корень из x |
| math.cos(X) | Косинус X (X указывается в радианах). |
| math.sin(X) | Синус X (X указывается в радианах). |
| math.tan(X) | Тангенс X (X указывается в радианах). |
| math.acos(X) | Арккосинус X. В радианах. |
| math.asin(X) | Арксинус X. В радианах. |
| math.atan(X) | Арктангенс X. В радианах |
| math.degrees(X) | Конвертирует радианы в градусы. |
| math.radians(X) | Конвертирует градусы в радианы. |

Полный список математических функций можно вывести командой:

import math

>>> math

<module 'math' (built-in)>

>>> dir(math)

Результаты работы программы представлены ниже на рисунке 3:

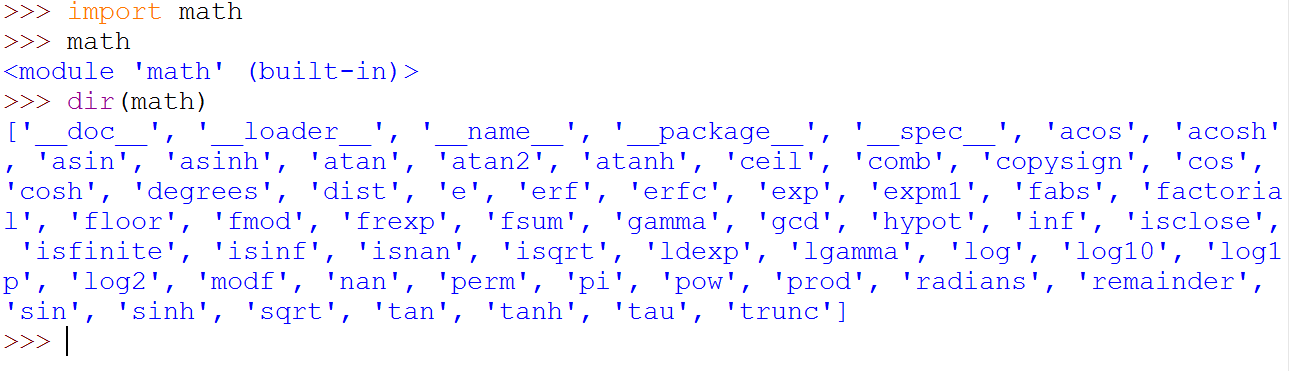


Рисунок 3 – Полный список математических функций языка Python

Приведем примеры использования данных математических функций.

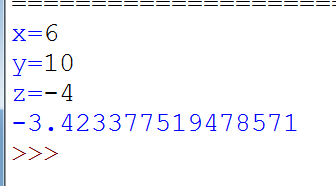
import math

x=float(input("x="))

y=float(input("y="))

z=float(input("z="))

print(math.sin(x)+math.exp(z)-math.sqrt(y))

Рисунок 4 – Результат работы программы

Также в языке Python есть возможно получения так называемых псевдо случайных чисел. Данные числа вычисляются по некоторой математической формуле, могут использоваться несколько алгоритмов (языке Python используется достаточно новый алгоритм «вихрь Мерсенна»).

В языке Python для получения таких чисел необходимо обратиться к модулю random.

Имеется возможность получать как целые, так и вещественные случайные числа:

* randint(a,b) – возвращает случайное целое число из отрезка [a; b];
* uniform(a,b) – возвращает случайное вещественное число из отрезка[a; b.

from random import randint

from random import uniform

k=randint( 4, 10)

m=uniform(1,200)

print(k)

print(m)

|  |  |
| --- | --- |
|  | Рисунок 5 – Результат работы программы |

*Материал для ученика*

Идентификатор – это уникальный признак объекта, позволяющий отличать объекты друг от друга, а значение – непосредственно информация, хранящаяся в памяти, которой управляет интерпретатор.

При инициализации переменной, на уровне интерпретатора, происходит следующее:

* создается целочисленный объект 5 (можно представить, что в этот момент создается ячейка и число 5 «кладется» в эту ячейку);
* данный объект имеет некоторый идентификатор, значение: 5, и тип: целое число;
* посредством оператора = создается ссылка между переменной b и целочисленным объектом 5 (переменная b ссылается на объект 5).

Базовый набор Python содержит следующие типы чисел:

* целые (int);
* вещественные (float) [с десятичной точкой];
* комплексные (complex) [состоят из действительной и мнимой части].

Целые числа (int)

Числа в Python 3 ничем не отличаются от обычных чисел. Они поддерживают набор самых обычных математических операций, представленных в таблице 1.

Таблица 1 – Операции над целыми числами в Python

|  |  |  |
| --- | --- | --- |
| **Обозначение операции** | **Описание** | **Пример** |
| x + y | Сложение | 5+9 |
| x – y | Вычитание | 6-2 |
| x \* y | Умножение | 4\*11 |
| x / y | Деление | 4/44 |
| x // y | Получение целой части от деления | 6//4 |
| x % y | Остаток от деления | 12%5 |
| -x | Смена знака числа | -7 |
| abs(x) | Модуль числа | abs(-16) |
| divmod(x, y) | Пара (x // y, x % y) | divmod(14, 3) |
| x \*\* y | Возведение в степень | 2\*\*8 |
| pow(x, y[, z]) | xy по модулю (если модуль задан) | pow(5,-2) |

Вещественные числа (float)

Еще такие числа называют числами с плавающей точкой. Это числа, содержащие точку (десятичный разделитель) или знак экспоненты. Числа типа float — неточны (из-за представления чисел с плавающей запятой в компьютере). Информацию о точности и внутреннем представлении float для вашей системы можно получить из sys.float\_info.

Все математические функции в языке Python объединены модуль math. Данный модель при необходимости нужно импортировать с помощью команды import:

import math

После подключения доступны математические функции языка Python.

В составе math присутствуют несколько известных математических констант:

Таблица 2 – Математические константы языка Python

|  |  |
| --- | --- |
| **Обозначение** | **Описание** |
| math.pi | Математическая константа π = 3.141592…. |
| math.e | Число Эйлера е=2,71828… |
| math.inf | Положительная бесконечность |

Опишем несколько популярных функций, входящих в состав модуля math.

Таблица 3 – Математические функции языка Python

|  |  |
| --- | --- |
| **Обозначение** | **Описание** |
| math.exp(x) | Экспонента числа ех |
| math.log(X, [А]) | Логарифм X по основанию А. Если А не указан, вычисляется натуральный логарифм |
| math.pow(X, n) | Возведение в степень Хn |
| math.sqrt(x) | Квадратный корень из x |
| math.cos(X) | Косинус X (X указывается в радианах). |
| math.sin(X) | Синус X (X указывается в радианах). |
| math.tan(X) | Тангенс X (X указывается в радианах). |
| math.acos(X) | Арккосинус X. В радианах. |
| math.asin(X) | Арксинус X. В радианах. |
| math.atan(X) | Арктангенс X. В радианах |
| math.degrees(X) | Конвертирует радианы в градусы. |
| math.radians(X) | Конвертирует градусы в радианы. |

Полный список математических функций можно вывести командой:

import math

>>> math

<module 'math' (built-in)>

>>> dir(math)

Также в языке Python есть возможно получения так называемых псевдо случайных чисел. Данные числа вычисляются по некоторой математической формуле, могут использоваться несколько алгоритмов (языке Python используется достаточно новый алгоритм «вихрь Мерсенна»).

В языке Python для получения таких чисел необходимо обратиться к модулю random.

Имеется возможность получать как целые, так и вещественные случайные числа:

* randint(a,b) – возвращает случайное целое число из отрезка [a; b];
* uniform(a,b) – возвращает случайное вещественное число из отрезка[a; b].

**Практикум 3.1 (1 час)**

Цель работы: Составление линейных программ, действия над целыми и вещественными числами.

    1. Составить программу вычисления значения функции *y*(*x*) = *x*2– 7*x*+ 8  для заданного с клавиатуры значения аргумента *х*.

    2. Составить программу вычисления значения выражения для вещественных значений *a*, *b*:

C:\Users\admin\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\2086BA4F.tmp

    3. Составить программу вычисления значения выражения для вещественных значений *a*, *b*:

C:\Users\admin\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\BE46C835.tmp

4. Составить программу, которая для двух целых чисел *m*, *n,* находит их среднее арифметическое.

5. Составить программу поиска координат вершины параболы y = *ax*2+ *bx*+ *c*.

**Практикум 3.2 (1 час)**

Цель работы: Составление и тестирование линейных программ, выполняющих действия над целыми и вещественными числами.

    1. Составить программу вычисления значения функции *y*(*x*) = (*x* + 1)2 + 3(*x*2 + 1)  для заданного с клавиатуры значения аргумента *х*.

    2. Составить программу вычисления расстояния между двумя точками, заданными на плоскости своими координатами.

    3. Составить программу вычисления площади боковой поверхности цилиндра, заданного радиусом основания и высотой.

    4. Составить программу вычисления площади равностороннего треугольника, заданного своей стороной.

    5. Составить программу вывода сумму цифр трехзначного числа, введенного пользователем.

*Тема 1.4. Оператор ветвления*

***Лекционное занятие (1 час)***

*Материал для учителя*

При составлении программ нам потребуются изучить три основные формы управления процессом выполнения программ. Согласно теории вычислительных систем, хороший язык программирования должен обеспечивать реализацию трех форм управления процессом выполнения программ.

1. Последовательность: выполнение последовательных операторов - с этим мы уже познакомились в более ранних курсах. Все наши предыдущие программы представляют собой некоторую последовательность операторов.
2. Выбор:

если такой и такой случай,

тогда сделать это

иначе сделать то.

1. Повторение:

пока (до тех пор, пока такой и такой случаи – делать это).

В данной лекции речь пойдет о реализации второй конструкции – выбора.

При решении задач важно реализовывать возможность выбора среди альтернативных операций на основе результатов проверки. В императивных языках программирования для этих целей используется оператор ветвления (условный оператор). В языке Python подобный оператор предусматривает не только возможность сделать выбор одной из двух альтернатив, но и предусматривает выполнение в зависимости от значения той или иной переменной одной из трех (или более) ветвей программы.

Общая форма оператора ветвления выглядит следующим образом:

if <условие1>:

оператор1

elif <условие2>:

оператор2

else:

оператор3

Части else и elif являются необаятельными. После части if указывается логическое условие, которое может быть истинным или ложным. Опишем случаи, когда выражение является истинным:

* Любое число, не равное 0, или непустой объект, а числа, равные 0, пустые объекты и значение None определяют ложь;
* Операции сравнения применяются к структурам данных рекурсивно;
* Операции сравнения возвращают True или False;
* Логические операторы and и or возвращают истинный или ложный объект-операнд.

Логическими операндами могут быть and (конъюнкция), or (дизъюнкция), not – отрицание. Равенство обозначается == (чтобы отличить знак равенства от оператора присвоения), а неравество !=. Также в Python можно записывать двойное условие, например 2<=a<=5, -10<v<=9.

Как видно из описания оператора ветвления, он может содержать в своем составе другие операторы, в том числе другие условные. Стоит обратить внимание, что после логического условия стоит двоеточие, для того чтобы показать, что далее идет блок выражений. Блок выражений записывается после отступа.

Рассмотрим работу оператора более подробно.

В самом простом случае оператор ветвления имеет вид:

if <условие1>:

оператор1

Блок-схема работы данного оператора представлена на рисунке 1:

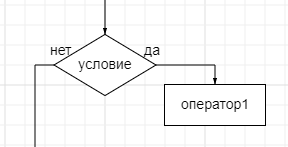


Рисунок 1 – Блок-схема условного оператора

В этом случае в случае истинности условия выполняется оператор1, а затем осуществляется выход из оператора (управление передается оператору, следующему за оператором if).

Приведем пример:

a = int(input())

if a < 0:

print('Ниже')

|  |  |
| --- | --- |
|  | Рисунок 2 – Результат работы программы |

В данном примере в качестве условия используется сравнение a<0. Если оно выполняется, то на экран выводится «Ниже». Если же условие ложно, то программа ничего не выполняет.

a = int(input())

if (a < 0) and(a>=-3):

print('Ниже')

|  |  |
| --- | --- |
|  | Рисунок 3– Результат работы программы |

***Практика4.1 (1 час)***

Цель работы: Составление и тестирование программ, в основе которых лежат условные алгоритмы. Получение навыков работы с оператором ветвления if в Python.

    1. Составить программу, которая проверяет, является ли данный прямоугольник квадратом.

    2. Составьте программу, определяющую, пройдет ли график функции y=5x2-7x+2 через заданную точку с координатами (а,b).

    3. Составьте программу, проверяющую, что введенное число является четным или нечетным.

    4. Определить, могут ли три данных числа быть сторонами треугольника.

    5. Проверить, принадлежит ли точка с координатами (x,y) осям координат.

***Лекционное занятие (1 час, продолжение)***

*Материал для учителя*

Рассмотрим далее более сложный вид оператора ветвления:

if <условие1>:

опертор1

else:

оператор2

Обратите внимание на порядок отступов в формате оператора!

Блок-схема работы данного оператора представлена на рисунке 4:



Рисунок 4 – Блок-схема условного оператора

Если условие истинно, то далее выполняется оператор1, а в противном случае (если условие ложно), то выполняется оператор2. А далее управление переходит к оператору, которое следует за оператором ветвления.

Приведём примеры работы более сложной формы оператор ветвления.

a = int(input())

b = int(input())

if a+b>15:

print('Yes')

else:

print('No')

|  |  |
| --- | --- |
|  | Рисунок 5 – Результат работы программы |

В данном примере в случае истинности условия a+b>15 выполняется оператор print('Yes'), а в противном случае – print('No').

Обратите внимание, что после условия и после else можно указывать несколько операторов, но тогда все они идут после отступа! Для сравнения рассмотрим два примера.

a=10; b=15; c=3

if a+b>10:

print('Yes')

else:

print('No')

c=a+b

print(c)

Результат работы данной программы представлен на рисунке 6:

|  |  |
| --- | --- |
|  | Рисунок 6 – Результат работы программы из примера |

a=10; b=15; c=3

if a+b>10:

print('Yes')

else:

print('No')

c=a+b

print(c)

|  |  |
| --- | --- |
|  | Рисунок 7– Результат работы программы из примера |

Как можно видеть, на рисунках 6-7 – результат работы программ отличается, так как строка c=a+b имеет различные отступы – соответственно, листинг сработал по-разному.

Рассмотрим еще один пример.

a=10

if a < -10:

print('Не долетел')

elif -10 <= a <= 10:

print('Попал')

else:

print('Перелетел')

Результат работы данной программы представлен на рисунке 7:

|  |  |
| --- | --- |
|  | Рисунок 8 – Результат работы программы из примера 4.2 |

***Практика4.2 (1 час)***

  Цель работы: Составление и тестирование программ, в основе которых лежат условные алгоритмы. Получение навыков работы с оператором ветвления if в Python.

    1. Составить программу, которая выводит стоимость покупки с учетом скидки. Скидка в 3% предоставляется, если стоимость покупки больше 500 р.

    2. Составить программу, которая запрашивает у пользователя целое число. Проверить является ли это число четырехзначным, если является, то выведите строку "Успешно", иначе "Неудача".

    3. Составить программу нахождения наибольшего среди трех целых чисел.

    4. Составить программу, которая запрашивает два целых числа. Уменьшить второе число в пять раз, если оно больше первого по абсолютной величине. В противном случае – заменить остатком от деления на 5.

    5. Составить программу, которая запрашивает целое трехзначное число. Если сумма цифр числа четная, то увеличить число на 2, в противном случае уменьшить на 2.

***Лекционное занятие (1 час)(продолжение)***

*Материал для учителя*

Третья форма оператора ветвления выглядит следующим образом:

if <условие1>:

опертор1

elif <условие2>:

опертор2

else:

опертор3

Блок-схема работы данного оператора представлена на рисунке 1:

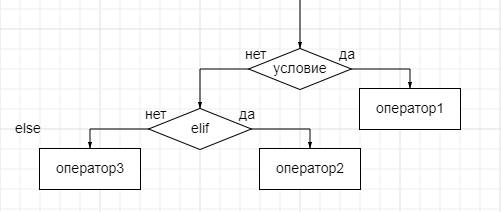


Рисунок 1 – Блок-схема условного оператора

При использовании данной формы можно проводить проверку нескольких условий – после if и после elif. Оператор после else выполняется в том случае, если не выполнилось условие 2 после части elif.

Рассмотрим пример использования данной формы оператора if.

a=10

if a < -5:

print('Да')

elif -5 <= a <= 5:

print('Возможно')

else:

print('Нет')

Результат работы представлен на рисунке 2:

|  |  |
| --- | --- |
|  | Рисунок 2 – Результат работы программы |

Написать программу вычисления стоимости покупки с учетом скидки. Скидка в 5% предоставляется в том случае, если стоимость покупки превысила 1000, а в 10% – если стоимость выше 3000 р.

Листинг программы представлен ниже:

s=int(input())

if s<1000:

print(s)

elif 1000<=s<3000:

print(s\*0.97)

else:

print(s\*0.95)

Результаты работы программы представлены ниже на рисунке 3:

|  |  |
| --- | --- |
|  | Рисунок 3 – Результат работы программы |

Еще раз обратимся к синтаксическим правилам языка Python.

В языке Python отсутствуют фигурные скобки, как в языке С/С++, или разделители «begin/end», как в языке Pascal, окружающие блоки программного кода. Вместо этого принадлежность операторов к вложенному блоку определяется по величине отступов. Так же операторы в языке Python обычно не завершаются точкой с запятой; обычно признаком конца инструкции служит конец строки с этой инструкцией.

Все составные операторы в языке Python оформляются одинаково: строка с заголовком завершается двоеточием, далее следуют одна или более вложенных инструкций, обычно с отступом относительно заголовка. Эти инструкции с отступами называются блоком (или иногда набором). Интерпретатор автоматически определяет границы блоков по величине отступов – т. е. по ширине пустого пространства слева от программного кода. Все инструкции, смещенные вправо на одинаковое расстояние, принадлежат к одному и тому же блоку кода.

В инструкции if предложения elif и else являются не только частями инструкции if, но и заголовками с собственными вложенными блоками. На Рисунок представлена схема с отступами для оператора if.

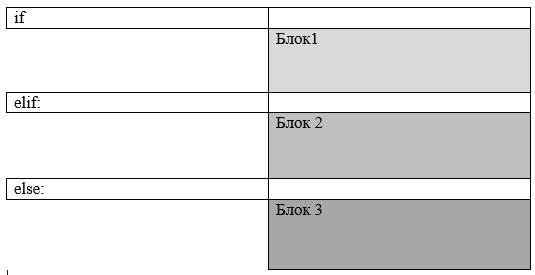


Рисунок 4 – Схема вложенных блоков

Примечание: в документации Python группа операторов, определяемая отступом, часто называется набором. Термины блок и набор взаимозаменяемы.

Также стоит отметить, что допустимо написать целое утверждение if на одной строке. Будет выглядеть так:

if <условие>: <оператор>

В одной строке может быть даже несколько <оператор>, разделенных точкой с запятой:

if <условие>: <оператор1>; <оператор2>; ...; <операторn>

Пример использования данного оператора:

x = 2

if x == 2: print('one'); print('two'); print('three')

Результат работы представлен ниже (рис.5).

|  |  |
| --- | --- |
|  | Рисунок 5 – Результат работы программы |

Часть else также может присутствовать:

x = 2

if x == 1: print('one'); print('two'); print('three')

else:

print('four')

|  |  |
| --- | --- |
|  | Рисунок 6 – Результат работы программы |

***Практика4.3 (1 час)***

Цель работы: Составление и тестирование программ, в основе которых лежат условные алгоритмы. Получение навыков работы с оператором ветвления if в Python.

    1. Составить программу, определяющую вид треугольника (равносторонний, равнобедренный, разносторонний) по заданным длинам сторон.

    2. Составить программу, которая определяет является ли введенный год високосным. В високосном году 366 дней, а в обычном – 365. Номер високосного года, делится на 4, за исключением номеров тех годов, которые делятся на 100 и не делятся на 400 (например, годы 300, 1300 и 1900 не являются високосными, а 1200 и 2000 – являются).

    3. Составить программу, которая по номеру введенного года определяет время года, к которому он принадлежит.

    4. Составить программу, которая запрашивает время в часах. Если введенное значение относится к диапазону от 7 до 10, то программа выводит – Пора вставать, в противном случае – Ты проспал.

    5. Составить программу, которая запрашивает ввод трех вещественных чисел x, y, z. Определить, существует ли прямоугольный треугольник со сторонами x, y, z. Если – да, вычислить его площадь.

***Лекционное занятие (1 час)(продолжение)***

*Материал для учителя*

Внутри условных инструкций можно использовать любые инструкции языка Python, в том числе и условную инструкцию. Таким образом, использовать операторы if-else можно внутри других инструкций if или if-else.

Синтаксически конструкция выглядит следующим образом:

1. Сначала записывается часть if с условным выражением, которое возвращает истину или ложь;
2. Затем может следовать одна или несколько необязательных частей elif (в других языках вы могли встречать else if);
3. Завершается же запись этого составного оператора также необязательной частью else.

Вложенная конструкция ветвления выглядит следующим образом:

if условие1:

блок кода

else:

if условие2:

блок кода

else:

if условие3:

блок кода

...

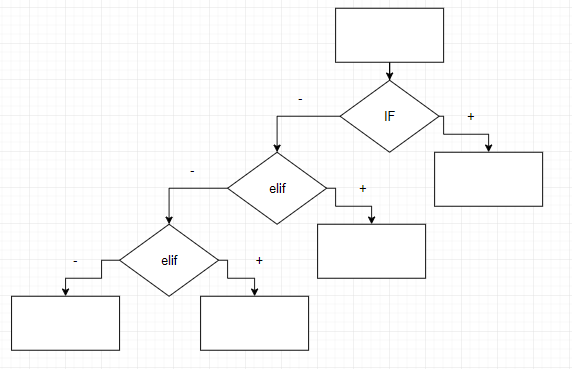


Рисунок 1 – Блок-схема множественного ветвления

Также стоит отметить, что в язык Python встроена возможность настоящего множественного ветвления на одном уровне вложенности, которое реализуется с помощью веток elif. Обратите внимание, в конце, после всех elif, может использоваться одна ветка else для обработки случаев, не попавших в условия ветки if и всех elif.

Рассмотрим работу с множественными условиями на примере.

a = int(input("Введите текущий лимит: "))

b= int(input("Введите кредитный рейтинг: "))

if b> 70:

if a > 150:

print("Поздравляем, вам выдан кредит")

else:

print("Извините, вы не имеете права на кредит")

Результат работы программы представлен на рис. 2.

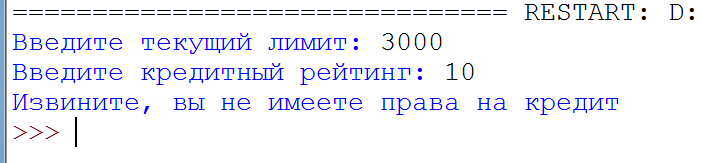


Рисунок 2 – Результат работы программы

Также язык Python поддерживает дополнительный метод принятия решений, называемую условным выражением. (Он также упоминается как условный оператор или тернарный оператор в различных местах документации Python).

В своей простейшей форме синтаксис условного выражения выглядит так:

<оператор1> if <условие> else <оператор2>

Это отличается от форм операторов if, перечисленных в предыдущих лекциях, потому что это не управляющая структура направляет поток выполнения программы. Он действует скорее, как оператор, определяющий выражение. В приведенном выше примере сначала вычисляется <условие>. Если истина, то выражение вычисляется как <оператор1>. Если ложь, то выражение вычисляется как <оператор2>.

Обратите внимание на не очевидный порядок: сначала вычисляется среднее выражение, и на основе этого результата возвращается одно из выражений на концах. Рассмотрим пример.

age = 12; s=''

s = 'minor' if age < 21 else 'adult'

print(s)

|  |  |
| --- | --- |
|  | Рисунок 3 – Результат работы программы |

Условие age<21 не выполнилось, поэтому переменная s получила значение 'minor'.

Запишем программу поиска наибольшего из двух целых чисел.

a=int(input())

b=int(input())

m = a if a > b else b

print(m)

|  |  |
| --- | --- |
|  | Рисунок 4 – Результат работы программы |

Данную задачу можно было б решить и стандартным оператором ветвления if:

a=int(input()); b=int(input())

if a>b: m=a

else: m=b

print(m)

Но условный оператор короче и, возможно, более читабельнее. Какой вариант использовать – зависит от решаемой задачи.

Условное выражение ведет себя как синтаксическое выражение. Его можно использовать как часть более длинного выражения. Стоит учитывать, что условное выражение имеет более низкий приоритет, чем практически все остальные операторы, поэтому для его группировки необходимы круглые скобки.

a=int(input()); b=int(input())

z = 1 + (a if a > b else b) + 2

print(z)

|  |  |
| --- | --- |
|  | Рисунок 5 – Результат работы программы |

***Практика4.4 (1 час)***

Цель работы: Составление и тестирование программ, в основе которых лежат условные алгоритмы. Получение навыков работы с оператором ветвления if в Python.

    1. Составить программу, которая запрашивает ввод трех целых чисел и выводит их на экран в порядке возрастания.

    2. Составить программу, которая для заданного n, 0<n≤100, рассматриваемого как возраст человека, вывести фразу типа  «Мне 20 лет», «Мне 21 год», «Мне 44 года».

    3. Составить программу, которая запрашивает действительные  положительные числа x, y, z. Выяснить, существует ли треугольник с длинами сторон x,y,z. Если треугольник существует, то ответить – является ли он остроугольным.

    4. Составить программу, которая для точки с координатами (x, y) определяет номер координатной четверти, к которой она принадлежит.

    5. Составить программу, которая запрашивает три стороны одного треугольника и три стороны другого треугольника. Определить, будут ли эти треугольники равновеликими, т. е. имеют ли они равные площади. Если это не так, то вывести «Ошибка».

*Тема 1.5. Цикл for и while*

***Лекционное занятие (1 час)***

*Материал для учителя*

Цикл в языке программирования представляет собой конструкцию, многократно выполняющую одну и ту же группу операторов. Число повторений циклов, или итераций, может быть задано заранее или зависеть от истинности того или иного условия. В реальной жизни постоянно применяются циклы, поэтому циклический алгоритм часто используются при решении задач по программированию.

В языке программирования Python может быть реализовано два вида цикла:

* С предусловием – цикл while;
* С параметром – цикл for.

Первый цикл, используемый в языке Python – цикл с параметром. Синтаксис данного цикла представлен ниже:

for <переменная> in <объект>:

<оператор1>

else:

<оператор2>

Блок-схема работы цикла выглядит следующим образом:

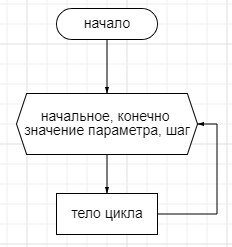


Рисунок 1 – Блок-схема цикла с параметром

Этот цикл проходится по любому итерируемому объекту (например строке или списку), и во время каждого прохода выполняет тело цикла заданное число раз. Для обращения к текущему элементу последовательности в теле цикла обычно используется переменная цикла, ее иногда называют управляющей переменной. В цикле for также используют необязательную часть else, которая работает точно так же, как и в циклах while, – она выполняется, если выход из цикла производится не инструкцией break (о которой будет рассказано ниже).

Часто для организации работы цикла с параметром for используется функция range.

Функция range() что возвращает последовательность чисел, регулируемую количеством переданных в неё аргументов. Возможны следующие варианты обращения к данной функции:

* range(finish);
* range(start, finish);
* range(start, finish, step).

Здесь start – это первый элемент последовательности (включительно), finish – последний (не включительно), а step – разность между следующим и предыдущим членами последовательности.

Например, range(5) возвращается последовательность 0, 1, 2, 3, 4. Вызов range(2,8) возвращается последовательность 2, 3, 4, 5, 6,7.

Рассмотрим примеры организации работы цикла с параметром.

for a in range(10):

print(a)

|  |  |
| --- | --- |
|  | Рисунок 2 – Результат работы программы |

В данном примере цикл выводит на экран последовательность числе от 0 до 9 включительно.

for c in range(2, 12, 3):

print(c, end=" ")

В данном примере выводится на экран последовательность числе от 2 до 12 с шагом 3.

|  |  |
| --- | --- |
|  | Рисунок 3 – Результат работы программы |

Рассмотрим еще несколько примеров цикла с параметром. Значения, используемые в функции range можно вводить с клавиатуры.

n=int(input())

for c in range(n):

print(c, end=" ")

|  |  |
| --- | --- |
|  | Рисунок 4 – Результат работы программы |

Также можно организовывать обратные циклы:

for c in range(12,5,-1):

print(c, end=" ")

|  |  |
| --- | --- |
|  | Рисунок 5 – Результат работы программы |

Обратите внимание, что указание в этом случае отрицательного шага – обязательно.

for c in range(15,7):

print(c, end=" ")

Такой фрагмент программы не вызовет ошибку, но на экран вывод не последует, так как цикл с параметром сращу закончится.

for c in range(15,7,-2):

print(c, end=" ")

|  |  |
| --- | --- |
|  | Рисунок 6 – Результат работы программы |

В данном фрагменте цикл выводит на экран значения с шагом -2.

***Практика 5.1(1 час)***

Цель работы: Составление и тестирование программ, в основе которых лежат циклические алгоритмы. Получение навыков работы с оператором цикла в Python.

1. Составить программу, которая выводит на экран таблицу квадратов первых десяти положительных чисел.

2. Составить программу, которая находит сумму пяти чисел, введенных с клавиатуры.

3. Составить программу, которая находит факториал числа, введенного с клавиатуры.

4. Составить программу, которая выводит нечетные числа в диапазона от 0 до n (диапазон – это последовательность чисел от 0 до n).

5. Составить программу, которая выводит числа, делящиеся на три без остатка, в диапазоне от 0 до n.

*Тема 1.5. Цикл for и while*

***Лекционное занятие (1 час)(продолжение)***

*Материал для учителя*

Цикл while также используется для повторения частей кода, но вместо зацикливания на n количество раз, он выполняет работу до тех пор, пока не достигнет определенного условия.

Цикл while является одним из самых часто используемых и самых универсальных циклов в Python. Полный формат данного цикла представлен ниже:

while <условие>:

<оператор1>

else:

<оператор2>

Часть else является необязательной, она выполняется, когда управление передается за пределы цикла без использования инструкции break.

|  |  |
| --- | --- |
|  | Рисунок 1 – Блок-схема цикла с предусловием while |

Выполнение цикла while начинается с вычисления выражения. Если оно истинно (не равно false), выполняется оператор цикла. Если при первой проверке выражение равно false, цикл не выполнится ни разу. Тип выражения должен быть арифметическим или приводимым к нему. Если условие в цикле while никогда не будет ложным, то не будет причин остановки цикла и программа «зациклится». Чтобы этого не произошло, необходимо организовать момент выхода из цикла, то есть ложность выражения в заголовке. Так, например, изменяя значение какой-нибудь переменной в теле цикла, можно довести логическое выражение до ложности. Обратите внимание, что операторы тела цикла должны быть отделены отступом.

i = 3

while i < 11:

print(i); i+= 3

|  |  |
| --- | --- |
|  | Рисунок 2 – Результат работы программы |

В данном примере организован перебор значений переменной i с шагом 3.

Условие работы цикла i < 11. В теле цикла происходит изменение переменной i, поэтому цикл не будет бесконечным.

Данный цикл можно рассматривать как повторяющийся оператор if. Когда мы не знаем количество итераций, цикл while является наиболее эффективным.

Рассмотрим следующий пример.

d = 0

while d < 7:

print("Python"); d+= 1

В данном примере цикл выполняется, пока истинно условие d < 7, значение переменной а меняется в теле цикла.

|  |  |
| --- | --- |
|  | Рисунок 3 – Результат работы программы |

При работе с данным циклом возникает опасность создания так называемого бесконечного цикла, или зацикливания. Данная ситуация может возникнуть, когда управляющее условие в цикле остается тожественно истинным.

a=10

while a>2:

print(a); a+=2

В данном цикле управляющее условие имеет вид: a>2. Оно останется тождественно истинным, так как используемая в нем переменная «растет» в теле цикла a+=2.

|  |  |
| --- | --- |
|  | Рисунок 4 – Результат работы программы |

В большинстве случаев, бесконечные циклы появляются из-за логических ошибок программиста (например, когда условие цикла while при любых вариантах равно True). Поэтому следует внимательно следить за условием, при котором цикл будет завершаться.

Однако вы некоторых случая бесконечный цикл делают намерено:

* Если нужно производить какие-то действия с интервалом, и выходить из цикла лишь в том случае, когда внутри тела "зашито" условие выхода. Пример: функция, которая возвращает connection базы данных. Если связь с базой данных отсутствует, соединение будет пытаться (в цикле) установиться до тех пор, пока не установится.
* Если вы пишете скрипт, который продолжительное время висит как процесс в системе и периодически производит какие-то действия. В таком случае остановкой цикла будет прерывание работы программы. Пример: скрипт, который раз в 10 минут "пингует" IP адреса и пишет в лог отчет о доступности этих адресов.

В бесконечных циклах рекомендуется ставить таймаут выполнения после каждой итерации, иначе вы очень сильно нагрузите CPU.

***Практика 5.2(1 час)***

Цель работы: Составление и тестирование программ, в основе которых лежат циклические алгоритмы. Получение навыков работы с оператором цикла в Python.

    1. Составить программу, которая выводит на экран последовательность от 1 до n с шагом 3.

    2. Составить программу, которая выводит на экран сумму цифр натурального числа.

    3. Составить программу, которая выводит на экран количество цифр натурального числа.

    4. Составить программу, которая запрашивает два положительных числа A и B (A > B). На отрезке длины A размещено максимально возможное количество отрезков длины B (без наложений). Не используя операции умножения и деления, найдите длину незанятой части отрезка A.

    5. Составить программу, которая запрашивает целое число N (>0). Найти наименьшее целое положительное число K, квадрат которого превосходит N: K2 > N.

*Тема 1.5. Цикл for и while (продолжение)*

***Лекционное занятие (1 час)***

Говоря про работу циклов в Python, следует упомянуть операторы continue, break, else.

Оператор continue используется для переходя на следующую итерацию цикла, пропуская следующего после него операторы тела цикла.

for i in range(15):

if i == 5:

continue

print(i \*\*2, end=' ')

|  |  |
| --- | --- |
|  | Рисунок 1 – Результат работы программы |

Здесь при равенстве переменной i==5 используется оператор continue, в результате чего пропускается оператор print(i \*\* 2, end=' '). Поэтому число 25 не выводится на экран.

Оператор break используется для организации немедленного выхода из цикла. То есть происходит досрочное завершение работы цикла.

for i in range(15):

if i == 10:

break

print(i \*\* 2, end=' ')

|  |  |
| --- | --- |
|  | Рисунок 2 – Результат работы программы |

В данном примере при равенстве переменной i==10 используется оператор break, в результате чего происходит завершение работы цикла. То есть последнее значение, рассмотренное в теле цикла будет i=9.

В случае вложенных циклов, оператор break завершает работу только того цикла, внутри которого он был вызван:

Рассмотрим пример.

i = 10; j = 20

while i < 15:

while j < 25:

if j == 23:

break

print('J', j)

j += 1

print('I', i)

i += 1

|  |  |
| --- | --- |
|  | Рисунок 3 – Результат работы программы  На рисунке хорошо видно, что здесь видно, что внутренний цикл прерывается, но внешний продолжает работу.  В Python не существует конструкций, которая прерывала бы сразу несколько циклов, но можно использовать дополнительные инструменты, например, флаговые переменные |

break\_the\_loop = False

for i in range(1,15):

for j in range(4,10):

print("compare: ", i, j)

if i == j:

print(f"found {i}")

break\_the\_loop = True

break

if break\_the\_loop:

break

|  |  |  |
| --- | --- | --- |
|  | Рисунок 4 – Результат работы программы  В данном примере переменная break\_the\_loop выполняет роль так называемого флага, она принимает значение True, в результате чего прерывается и второй цикл.  Оператор else используется для проверки, был ли произведен выход из цикла инструкцией break, или же цикл завершился образом. | |
| for i in range(10):  if i == 15:  break  print(i \* 2, end=' ')  else:  print("Not Error") | | Рисунок 4 – Результат работы программы  В данном примере после вывода на экран последовательности от 0 до 18 на экран также выводится строка "Not Error", так как оператор break не сработал. |

Заметим, что если цикл содержит оператор break, то оператор else не будет выполнен.

for i in range(0,5):

print(i)

else:

print("The end")

|  |  |
| --- | --- |
|  | Рисунок 5 – Результат работы программы  Цикл for полностью исчерпал себя, так как нет прерывания.  В реальной практике, else в циклах применяется нечасто. Такая конструкция отлично сработает, когда будет необходимо проверить факт выполнения всех итераций цикла. |

***Практика 5.3(1 час)***

Цель работы: Составление и тестирование программ, в основе которых лежат циклические алгоритмы. Получение навыков работы с оператором цикла в Python.

1. Составить программу проверки введенного с клавиатуры целого числа на «простоту».

2. Составить программу вывода на экран всех 3-значных чисел x, где (x+1) четное, а (x+2) кратно 3-м.

3. Составить программу вывода на экран всех 3-значных чисел, где сумма цифр равна заданному n.

4. Составить программу поиска среди n введенных целых чисел наибольшее четное.

5. Составить программу, которая запрашивает последовательность из n целых чисел. Первое число в последовательности чётное. Найти сумму всех чётных чисел.

*Тема 1.5. Цикл for и while (продолжение)*

***Лекционное занятие (1 час)***

Также в языке Python возможно использование вложенных циклов, когда есть один внешний цикл и один или несколько вложенных. Глубина вложений не оговаривается – можно вкладывать любое количество циклов внутри цикла for или while.

Внутренний цикл выполняется n раз на каждой итерации внешнего цикла. Программа сначала выполняет внешний цикл. Внутри первой итерации внешнего цикла запускается внутренний, вложенный цикл. Затем программа возвращается обратно к началу внешнего цикла, завершает вторую итерацию и снова вызывает вложенный цикл. После завершения вложенного цикла программа возвращается в начало внешнего цикла. Это будет происходить до тех пор, пока последовательность не будет завершена или прервана (или пока какое-то выражение не приведёт к нарушению процесса).

|  |  |  |
| --- | --- | --- |
|  | | Рисунок 1 – Блок-схема вложенного цикла  Стоит отметить, что использование вложенных циклов может замедлить работу программы.  Приведем пример программ с вложенными циклами.  for i in range(6):  for j in range(1,4):  print(j, end=' ')  print()  Результат работы приведен ниже (рис. 2). |
|  | Рисунок 2 – Результат работы программы  Как видно на рисунке, программа выполнила 6 итераций внутреннего цикла, который выводит на экран значения от 1 до 3.  Приведем еще несколько примеров работы с циклами for и while.  Рассмотрим реализацию алгоритма Евклида. | |

Есть два варианта реализации данного алгоритма – с помощью получения остатков от деления и с помощью получения разности.

|  |  |
| --- | --- |
| Вариант 1  a=int(input()); b=int(input())  while a!=0 and b!=0:  if a>=b: a%=b  else: b%=a  print(a+b) | Вариант 2  a=int(input()); b=int(input())  while a!=b:  if a>b: a-=b  else: b-=a  print(a) |

|  |  |
| --- | --- |
|  | Рисунок 3 – Результат работы программы для первого варианта |
|  | Рисунок 4 – Результат работы программы для второго варианта  Стоит отметить, что модуле math Python есть функция gcd(), вычисляющая наибольший общий делитель двух чисел. |

Продолжим рассмотрение примеров.

На тренировке спортсмен ежедневно пробегает некоторую дистанцию, с каждым днем увеличивая ее на 10%. Составить программу, определяющую по расстоянию, преодоленному спортсменом в первый день тренировки длину дистанции на k-ый день.

n= float(input("Введите начальную дистанцию "))

k= int(input("Введите количество дней "))

for i in range (k): n+=n\*0.1

print("конечная дистанция ", n)

|  |  |
| --- | --- |
|  | Рисунок 5 – Результат работы программы |

В задаче используется цикл for, так как известно количество повторов цикла – k дней. Внутри цикла идет увеличение переменной n, обозначающей длину дистанции, n+=n\*0.1.

Переведите введенное пользователем десятичное число в двоичное. Известно, что число меньше 256.

dec=int(input("Введите десятичное число "))

v=128

for i in range(1,9):

if dec>=v:

print('1', end='')

dec=dec-v

else:

print('0', end='')

v=v//2

|  |  |
| --- | --- |
|  | Рисунок 6 – Результат работы программы |

В данном примере с помощью переменной v задается вес старшего разряда, так как по условию число должно быть меньше 256, то старший разряд будет иметь значение 128. Для перевода десятичного числа из него каждый раз вычитается старший разряд, если это возможно, затем значение разряда уменьшается.

Разложить натуральное число на простые множители.

k=int(input("Введите число "))

print(k,'= '); l=2

while not(k==1):

if k%l==0: k=k/l; print(l, end=' ')

else: l+=1

|  |  |
| --- | --- |
|  | Рисунок 7 – Результат работы программы  В данном примере первое простое число l=2. В цикле while введенное k несколько раз делится на потенциальный простой делитель l, если целочисленное деление не может быть выполнено, то ищется следующий простой делитель. |

***Практика 5.4(1 час)***

Цель работы: Составление и тестирование программ, в основе которых лежат циклические алгоритмы. Получение навыков работы с оператором цикла в Python.

    1. Составить программу, которая по данному натуральному n ≤ 30 выведет лесенку из n ступенек, i-я ступенька состоит из чисел от 1 до i без пробелов.

    2. Составить программу, которая среди n введенных с клавиатуры целых чисел, находит число с наибольшей суммой цифр.

    3. Составить программу, которая промежутке натуральных чисел выводит те, количество делителей у которых не меньше введенного значения.

    4. Составить программу, которая выводит все совершенные числа до 1000. Совершенное число – это такое число, которое равно сумме всех своих делителей, кроме себя самого.

    5. Составить программу, которая решает следующую задачу. Плата за быка 10 рублей, за корову – 5 рублей, за телёнка – полтинник (0,5 рубля). Определить, можно ли ровно на M  рублей купить ровно N  голов скота. Если это возможно, вывести "yes", в противном случае – "no".

***Практика 5.5(2 часа)***

Цель работы: Составление и тестирование программ, в основе которых лежат циклические алгоритмы. Получение навыков работы с оператором цикла в Python.

    1. Составить программу, которая запрашивает ввод натурального числа q (0<q≤1000). Проверить, может ли оно являться периметром прямоугольного треугольника, стороны которого выражены натуральными числами.

    2. Составить программу, которая сокращает дробь m/n.

    3. Составить программу, которая выводит количество натуральных чисел взаимно простых с m (2  ≤ m ≤ 1000), меньших заданного числа n (2 ≤ n ≤ 1000). Два натуральных числа называют взаимно простыми, если их наибольший общий делитель равен 1.

    4. Составить программу, которая запрашивает два натуральных числа a и b (1≤ a ≤ b≤ 1000 ). Получить количество всех простых чисел, входящих в диапазон [a; b].

    5. Составить программу, которая просматривает в возрастающем порядке все трёхзначные числа. Вывести на экран те числа, все цифры у которых различны.