Урок №№ 1-2

Язык программирования (англ. Programming language) — это искусственный язык, созданный для разработки программ, предназначенных для выполнения на компьютере.

Компьютерная программа (англ. Computer program) — это последовательность команд (инструкций), которые обеспечивает реализацию на компьютере конкретного алгоритма.

Команда (инструкция) — это указание, которое определяет, какое действие (операцию) следует выполнять.

По ориентации на класс задач языки программирования делятся на универсальные и специализированные.

Универсальные языки предназначены для решения широкого класса задач. К таким языкам относятся PL/1, Algol, Pascal, C и др. Особым классом универсальных языков является визуальные среды программирования: VisualBasic, Delphi и др.

Специализированные языки учитывают специфику предметной области. В настоящее время существуют десятки специализированных языков программирования, например, языки веб-программирования, языки скриптов и др. Язык скриптов используется для создания небольших вспомогательных программ, например, JavaScript.

Составляющие части языка программирования.

Любой язык программирования высокого уровня, как и любой другой язык, имеет основные составляющие:

1. Алфавит - набор символов, из которых образуются команды программы и другие конструкции языка. Большинство из них содержит английские буквы, цифры, знаки арифметических операций (+, *, -, /), знаки отношений (больше, равно и др.), синтаксические знаки (точка, точка с запятой и др.).

2. Синтаксис - совокупность правил записи команд и других конструкций языка. Нарушение правил синтаксиса определяется автоматически, о чем программист получает сообщение.

3. Семантика - совокупность правил толкования и выполнения конструкций языка программирования.

4. Словарь— определенное количество слов, правила употребления которых определены для этого языка и которые имеют строго определенное назначение. Такие слова называют зарезервированными (ключевыми), например, for, input, if, print.

Этапы решения задачи на ЭВМ. Работа по решению любой задачи с использованием компьютера включает в себя следующие шесть этапов:

1. Постановка задачи.
2. Формализация задачи.
3. Построение алгоритма.
4. Составление программы на языке программирования.
5. Отладка и тестирование программы.
6. Проведение расчетов и анализ полученных результатов.

Непосредственно к программированию из этого списка относятся п. 3 - 5.

На этапе постановки задачи следует четко определить, что дано и что требуется найти. Важно описать полный набор исходных данных, необходимых для решения задачи.

На этапе формализации чаще всего задача переводится на язык математических формул, уравнений, отношений и таблиц.

Третий этап — это построение алгоритма.

Первые три этапа — это работа без компьютера.

Последующие два этапа — это собственно программирование на определенном языке в определенной системе программирования.

На последнем — шестом — этапе разработанная программа уже используется в практических целях.

Понятие алгоритма

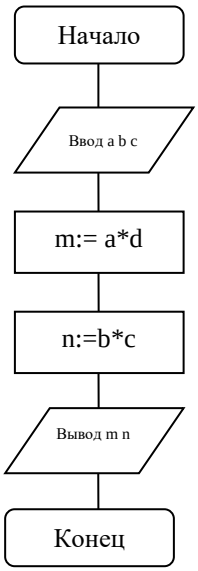
Применительно к ЭВМ алгоритм определяет вычислительный процесс, начинающийся с обработки исходных данных и направленный на получение результатов.

Выделяют следующие виды алгоритмов:

- линейный;
- алгоритм ветвления;
  циклический алгоритм;
  вспомогательный алгоритм, который можно использовать в других алгоритмах, указав только его имя.

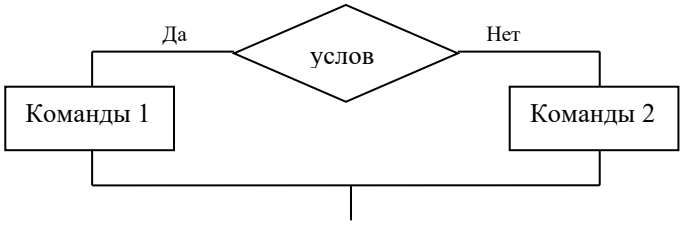
Линейный алгоритм – это алгоритм, в котором команды выполняются последовательно одна за другой, и состоит из команд присваивания, ввода, вывода и обращения к вспомогательным алгоритмам (подпрограммам).

Графическое описание будет выглядеть следующим образом:

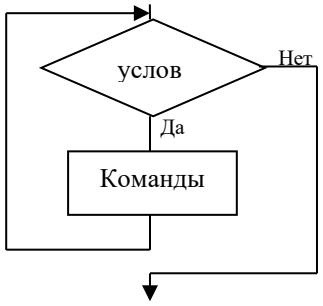


Алгоритм деления дробей имеет линейную структуру, т. е. в нем все команды выполняются в строго определенной последовательности, каждая по одному разу.

Алгоритм ветвления - это алгоритм, в котором в зависимости от условия выполняется либо одна, либо другая последовательность команд. В нем применяется следующий блок:



Циклический алгоритм - это алгоритм, в котором заданная последовательность команд выполняется пока справедливо некоторое условие. В нем применяется следующий блок:



Величины подразделяются на константы и переменные. Константа — неизменная величина, и в алгоритме она представляется собственным значением, например: 15, 34.7, k, True и др. Переменные величины могут изменять свои значения в ходе выполнения программы и представляются в алгоритме символическими именами — идентификаторами, например: X, S2, cod 15 и др.

Основные понятия структурного программирования.

В основе структурного программирования лежит следующая теорема: алгоритм для решения любой логической задачи можно составить только из структур Следование, Ветвление, Цикл.

Сложные алгоритмы состоят из соединенных между собой базовых структур. Соединение этих структур может выполняться двумя способами: последовательным и вложенным.

Структурный подход требует стандартного изображения блок- схем алгоритмов. Каждая базовая структура должна иметь один вход и один выход.

Возможны два подхода к построению алгоритма:

- сверху вниз — сначала строится основной алгоритм, а затем — вспомогательные;
- снизу-вверх — сначала составляются вспомогательные алгоритмы, а затем — основной.

Краткие сведение о языке Python

Язык программирования Python был создан к 1991 году голландцем Гвидо ван Россумом. После того, как Россум разработал язык, он выложил его в Интернет, где сообщество программистов присоединилось к его улучшению. Python активно развивается в настоящее время. Часто выходят новые версии. Существуют две поддерживаемые ветки: Python 2.x и Python 3.x. Здесь английской буквой "x" обозначается конкретный релиз. Между вторым и третьим Питоном есть небольшая разница.

Официальный сайт поддержки языка – http://python.org.

Python – интерпретируемый язык программирования. Это значит, что исходный код частями преобразуется в машинный в процессе его чтения специальной программой – интерпретатором.

Python характеризуется ясным синтаксисом. Читать код на нем легче, чем на других языках программирования, т. к. в Питоне мало используются такие вспомогательные синтаксические элементы как скобки, точки с запятыми. С другой стороны, правила языка заставляют программистов делать отступы для обозначения вложенных конструкций.

Python – это полноценный во многом универсальный язык программирования, используемый в различных сферах. Основная, но не единственная, поддерживаемая им парадигма, – объектно-ориентированное программирование.

Интерактивный режим

Это удобно, когда изучаешь особенности языка или тестирует какую-нибудь небольшую часть кода. Ведь если работать на компилируемом языке, то пришлось бы сначала создать файл с кодом на исходном языке программирования, затем передать его компилятору, получить от него исполняемый файл и только потом выполнить программу и оценить результат.

Создание скриптов

Несмотря на удобства интерактивного режима, чаще всего необходимо сохранить исходный программный код для последующего выполнения и использования. В таком случае подготавливаются файлы, которые передаются затем интерпретатору на исполнение. Файлы с кодом на Python обычно имеют расширение .ру.

Подготовить ответы на следующие вопросы:

- Охарактеризовать языки программирования низкого уровня.
- Охарактеризовать языки программирования высокого уровня.
- Охарактеризовать структурированные языки программирования.
- Охарактеризовать модульное программирования.
- Охарактеризовать ООП.
- Перечислить составляющие части языка программирования.
- Дать понятие системы программирования.
- Назначение библиотеки подпрограмм.
- Назначение компоновщика.
- Понятие компилятора.
- Понятие интерпретатора.
- Назначение транслятора.
- Назначение отладчика.
- Перечислите этапы решения задачи на ЭВМ.
- Что происходит на этапе постановки задачи? Почему важен этот этап?
- Дать понятие алгоритма.
- Какова особенность линейных алгоритмов.
- Понятие алгоритма ветвления.
- Понятие циклического алгоритма.
- Что понимается под данными?
- Перечислить свойства любой величины.
- Перечислить типы данных.
- Основная теорема структурного программирования.
- О языке Пайтон

Урок № 4

Арифметическое выражение. Содержит арифметические операции, функции, операнды, круглые скобки и др.

Для верной записи арифметических выражений следует соблюдать определенные правила.

1. Все символы писать в строку, т.е. на одном уровне. Проставлять все знаки операций, не пропуская знак «*».
2. Не допускать записи двух знаков операций подряд, т. е. нельзя писать A + -B, следует писать A + (-B).

3. Операции с более высоким приоритетом выполняют раньше операций с меньшим приоритетом. Порядок убывания приоритетов операций следующий:

- **;
- унарная операция смены знака (-);
- /, *, %, //;
- +, -;
- <, >, <=, >=
- ==, !=;
- not, and, or

4. Несколько записанных подряд операций с одинаковым приоритетом выполняются последовательно слева направо.

5. Часть выражения, заключенная в скобки, вычисляется в первую очередь. (Например, в выражении (A + B) * (C - D) умножение производится после сложения и вычитания.)

6. Не следует записывать выражения, не имеющие математического смысла, например: деление на нуль, логарифм отрицательного числа и т.п.

Действие логических операций and, or и not определяется с помощью таблиц истинности. В этих таблицах показывается, как зависит значение логических выражений A and B, A or B, not A от значений логических операндов A и B.

Таблица истинности операций and и or

A	B	A and B	A or B
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Таблица истинности операции not

A	not A
True	False
False	True

Логическое выражение	Описание
(x>=5) and (x<=10)	Истинно когда x принимает значения от 5 до 10
(x=3) or (x=5) or (x=7)	Истинно когда x принимает одно их трех значений: 3, 5 или 7
Not (x=5)	Истинно когда x <> 5
(x=3) and (x=5)	Всегда ложно

Урок № 6

Форматированный вывод строк

Форматирование строк не имеет отношения к print(), но обычно используется именно в сочетании с функцией print().

Форматирование может выполняться в так называемом старом стиле (Си-версия) или с помощью строкового метода format.

```
# C - стиль
pupil = "Ben"
old = 16
grade = 9.2
print("It ' s  %s, %d. Level: %.1f" % (pupil, old, grade))
It's Ben, 16. Level: 9.2
```

Здесь вместо трех комбинаций символов %s, %d, %f подставляются значения переменных pupil, old, grade. Буквы s, d, f обозначают типы данных – строку, целое число, вещественное число. Если бы требовалось подставить три строки, то во всех случаях использовалось бы сочетание %s.

.1 – задает количество знаков после запятой.

```
# метод format()

print("This is a {0}. It's {1}.".format("ball", "red"))
This is a ball. It's red.

print("This is a {0}. It's {1}.".format("cat", "white"))
This is a cat. It's white.

print("This is a {0}. It's {1} {2}.".format(1, "a", "number"))
This is a 1. It's a number.
```

В строке в фигурных скобках указаны номера данных, которые будут сюда подставлены. Далее к строке применяется метод format(). В его скобках указываются сами данные (можно использовать переменные). На нулевое место подставится первый аргумент метода format(), на место с номером 1 – второй и т. д.

```
print("Тебя зовут {0}. Твоя фамилия {1}. Ты студент!". format(input(),input()))
print(f"Тебя зовут {input()}. Твоя фамилия {input()}. Ты студент!")
```

Если фигурные скобки исходной строки пусты, то подстановка аргументов идет согласно порядку их следования. Если в фигурных скобках строки указаны индексы аргументов, порядок подстановки может быть изменен:

```
#1
>>> size1 = "length - {}, width - {}, height - {}"
>>> size1.format(3, 6, 2.3)
'length - 3, width - 6, height — 2.3'

#2
>>> size2 = "height - {2}, length - {0}, width - {1}"
>>> size2.format(3, 6, 2.3)
'height - 2.3, length - 3, width - 6'

#3
>>> n = 20
>>> m = 25
>>> prod = n * m
>>> print(f'Произведение {n} на {m} равно {prod}')
Произведение 20 на 25 равно 500
```

Кроме того, аргументы могут передаваться по слову- ключу:

```
>>> info = "This is a {subj}. It's {prop}."
>>> info.format(subj="table", prop="small")
"This is a table. It's small."
```

Урок № 7

Раздаточный материал № 22

```
#1
if n < 100:
    b = n + a #отступ является обязательным, т.к. формирует тело условного
    оператора
print(b) # оператор print не является телом условного оператора

#2
tovar1 = 50
tovar2 = 32
if tovar1 + tovar2 > 9 :
    print("99 рублей недостаточно")
else:
    print("Чек оплачен")

#3

a = 5 > 0 # подвыражение 5 > 0 выполнится первым, после чего его результат будет
        присвоен переменной a
if a:
    print(a)

if a > 0 and a < b:
    print(b - a)

if 0 < a < b:
    print(b - a)
```

Множественное ветвление предполагает выбор больше, чем из двух путей, например, из трех, четырех или даже пяти.

Раздаточный материал № 23

```
old = int(input('Ваш возраст: '))
print('Рекомендовано:', end=' ')
if 3 <= old < 6:
    print('"Заяц в лабиринте"')
elif 6 <= old < 12:
    print('"Марсианин"')
elif 12 <= old < 16:
    print('"Загадочный остров"')
elif 16 <= old:
    print('"Поток сознания"')
```

Урок № 18

Функция в программировании представляет собой обособленный участок кода, который можно вызывать, обратившись к нему по имени, которым он был назван. При вызове происходит выполнение команд тела функции.

Функции можно сравнить с небольшими программами, которые сами по себе, т. е. автономно, не исполняются, а встраиваются в обычную программу. Нередко их так и называют – подпрограммы. Функции также при необходимости могут получать и

возвращать данные. Только обычно они их получают не с ввода с клавиатуры, файла и др., а из вызывающей программы. Сюда же они возвращают результат своей работы.

Например, print(), input(), int() – это тоже функции. Код их тела нам не виден, он где- то "спрятан внутри языка". Нам же предоставляется только интерфейс – имя функции.

С другой стороны, программист всегда может определять свои функции. Их называют пользовательскими.

В языке программирования Python функции определяются с помощью оператора def.

Раздаточный материал № 36

```
def countFish():
    a = int(input())
    b = int(input())
    print("Всего", a+b, "шт.")
```

Функция состоит из заголовка и тела. Заголовок оканчивается двоеточием и переходом на новую строку. Тело имеет отступ.

Ключевое слово def сообщает интерпретатору, что перед ним определение функции. За def следует имя функции. Оно может быть любым, но со смыслом. После имени функции ставятся скобки. В приведенном примере они пустые. Это значит, что функция не принимает никакие данные из вызывающей ее программы. Однако она могла бы их принимать, и тогда в скобках были бы указаны параметры.

После двоеточия следует тело, содержащее инструкции, которые выполняются при вызове функции. Описание и вызов функции могут находиться в разных местах основной программы, но описание функции обязательно предшествует ее вызову. Можно определить функцию, но ни разу ее не вызвать.

Вызывается функция по имени со скобками. Если в функцию не предаются параметры, то скобки остаются пустыми. При вызове функции основной поток выполнения программы приостанавливается до момента завершения работы функции. Затем программа продолжает свою работу.

Раздаточный материал № 37

```
# программа выводит SOS

def CharS():
    print('S', end='')


def CharO():
    print('O', end='')


CharS()
CharO()
CharS()
```

Локальные и глобальные переменные

В программировании особое внимание уделяется концепции о локальных и глобальных переменных, а также связанное с ними представление об областях видимости. Соответственно, локальные переменные видны только в локальной области видимости, которой может выступать отдельно взятая функция. Глобальные переменные видны во всей

программе. "Видны" – значит, известны, доступны. К ним можно обратиться по имени и получить связанное с ними значение.

К глобальной переменной можно обратиться из локальной области видимости. К локальной переменной нельзя обратиться из глобальной области видимости, потому что локальная переменная существует только в момент выполнения тела функции. При выходе из нее, локальные переменные исчезают. Компьютерная память, которая под них отводилась, освобождается. Когда функция будет снова вызвана, локальные переменные будут созданы заново.

Раздаточный материал № 38

```
def rectangle():
    a = float(input("Ширина %s: " % figure)) # обращение к глобальной
    b = float(input("Высота %s: " % figure)) # переменной figure
    print("Площадь: %.2f" % (a*b))
def triangle():
    a = float(input("Основание %s: " % figure))
    h = float(input("Высота %s: " % figure))
    print("Площадь: %.2f" % (0.5 * a * h))
figure = input("1-прямоугольник, 2-треугольник: ")
if figure == '1':
    rectangle()
elif figure == '2':
    triangle()
```

Здесь пять переменных. Глобальной является только figure. Переменные a и b из функции rectangle(), а также a и h из triangle() – локальные. При этом локальные переменные с одним и тем же идентификатором a, но объявленные в разных функциях, – разные переменные.

Идентификаторы rectangle и triangle, хотя и не являются именами переменных, а представляют собой имена функций, также имеют область видимости. В данном случае она глобальная, так как функции объявлены непосредственно в основной ветке программы.

В приведенной программе к глобальной области видимости относятся заголовки объявлений функций, объявление и присваивание переменной figure, конструкция условного оператора.

К глобальным переменным можно обращаться из функций.

При построении функции необходимо иметь в виду, что изменять значения глобальных переменных внутри функции можно, но это плохая практика программирования. Вместо это необходимо предусмотреть возврат полученного в функции значения в глобальную область видимости. Это делает программу более понятной и позволяет избежать проблем, связанных с поиском причины изменения глобальной переменной.

В Пайтон внутри одной функции можно определить другую. Так же функция может возвращать значение в то место, откуда она была вызвана. Для этого используется оператор return. Как только интерпретатор встречает return, то он "забирает" значение, указанное после этой команды, и "уходит" из функции.

Раздаточный материал № 39

```
# В основной ветке программы вызывается функция cylinder(), которая
вычисляет площадь
# цилиндра. В теле cylinder() определена функция circle(), вычисляющая
площадь круга по
# формуле πr2. В теле cylinder() у пользователя спрашивается, хочет ли он
получить только
# площадь боковой поверхности цилиндра, которая вычисляется по формуле
```

```
    2πrh, или полную
    # площадь цилиндра. В последнем случае к площади боковой поверхности
    цилиндра должен
    # добавляться удвоенный результат вычислений функции circle().

    SC = 0
    SQ = 0


    def cylinder():
        r = float(input('Введи радиус:  '))

        def circle():
            SC = 3.14 * r * 2
            return SC

        c = input('1 - площадь боковой поверхности цилиндра, 2 - полная
    площадь цилиндра:  ')
        if c == '1':
            print(circle())
        elif c == '2':
            h = float(input('Введи высоту:  '))
            SQ = 2 * 3.14 * r * h + 2 * circle()
            print(SQ)


    cylinder()
```

Если необходимо сохранить результаты, передаваемые функцией, то их необходимо присвоить переменной. Например,

Раздаточный материал № 40

Port = `cylinder()`

В функции может быть несколько операторов return. Но выполнится только один, до которого первым дойдет поток выполнения (например, ветка except).

В Питоне можно возвращать из функции несколько объектов, перечислив их через запятую после команды return:

Раздаточный материал № 41

```
def duble():
    width = float(input('Введи ширину:   '))
    height = float(input('Введи высоту:   '))
    ploch = width * height
    perim  = 2 * (width + height)
    return ploch, perim

g_ploch, g_perim = duble()
print('Полощадь прямоугольника:  ', g_ploch)
print('Периметр прямоугольника:  ', g_perim)
```

Перечисление значений через запятую создает объект типа «кортеж» (tuple). Когда же кортеж присваивается сразу нескольким переменным, то происходит сопоставление его элементов соответствующим в очереди переменным. Это называется распаковкой.

Таким образом, когда из функции возвращается несколько значений, на самом деле из нее возвращается один объект класса «кортеж». Перед возвратом эти несколько значений

упаковываются в кортеж. Если же после оператора return стоит только одна переменная или объект, то ее/его тип хранится как есть.

Распаковка не является обязательной.

Раздаточный материал № 42

```
print(duble())

Введи ширину:   10
Введи высоту:   20
(200.0, 60.0) – скобки говорят, что выводится кортеж
```

Параметры и аргументы функции

В программировании функции могут не только возвращать данные, но также принимать их, что реализуется с помощью параметров, которые указываются в скобках в заголовке функции (формальные параметры) через запятую. Количество параметров может быть любым.

Параметры представляют собой локальные переменные, которым присваиваются значения в момент вызова функции (фактические параметры). Конкретные значения, которые передаются в функцию при ее вызове, будем называть аргументами.

Передача данных по значению

Раздаточный материал № 43

```
def duble(a, b):
    ploch = a * b
    perim  = 2 * (a + b)
    return ploch, perim

width = float(input('Введи ширину:   '))
height = float(input('Введи высоту:   '))
g_ploch, g_perim = duble(width, height)
print('Площадь прямоугольника:  ', g_ploch)
print('Периметр прямоугольника:  ', g_perim)
```

Параметры `width, height` являются глобальными и при вызове функции являются фактическими параметрами. Параметры a и b являются формальными и при вызове функции им будут присвоены значения `width, height.` Имена формальных и фактических параметров могут не совпадать (это даже не желательно).

Изменение значений a и b в теле функции никак не скажется на значении переменных `width, height`. Они останутся прежними. Поэтому говорят, что в функцию данные передаются по значению.

При работе с параметрами необходимо учитывать:

1. Количество формальных и фактических параметров должно совпадать (a,b и `width, height`).
2. Последовательность передачи значений (a присвоится значение `width`, b присвоится значение `height`).

В Python у функций бывают параметры, которым уже присвоено значение по-умолчанию. В таком случае, при вызове можно не передавать соответствующие этим параметрам аргументы. Хотя можно и передать. Тогда значение по умолчанию заменится на переданное.

Раздаточный материал № 44

```
# 1
def duble(a, b=20):
    ploch = a * b
    perim  = 2 * (a + b)
    return ploch, perim

width = float(input('Введи ширину:   '))
g_ploch, g_perim = duble(width)
print('Площадь прямоугольника:  ', g_ploch)
print('Периметр прямоугольника:  ', g_perim)

# 2
def duble(a, b=20):
    ploch = a * b
    perim  = 2 * (a + b)
    return ploch, perim

width = float(input('Введи ширину:   '))
height = float(input('Введи высоту:   '))
g_ploch, g_perim = duble(width, height)
print('Площадь прямоугольника:  ', g_ploch)
print('Периметр прямоугольника:  ', g_perim)
```

При вызове функции, можно явно указывать, какое значение соответствует какому параметру. В этом случае их порядок не играет роли.

Раздаточный материал № 45

```
def duble(a, b):
    ploch = a * b
    perim = 2 * (a + b)
    return ploch, perim


g_ploch, g_perim = duble(b=20, a=10)
print('Площадь прямоугольника:  ', g_ploch)
print('Периметр прямоугольника:  ', g_perim)
```

Функция может быть определена так, что в нее можно передать множество параметров:

Раздаточный материал № 46

```
def oneOrMany(*a):
    print(a)

oneOrMany(1)
oneOrMany('1', 1, 2, 'abc')
oneOrMany()
```

Результат:
(1,)
('1', 1, 2, 'abc')
()