Программирование. Подготовка к экзамену

От автора

Куда скидывать найденные очепятки и печенюшки Вы, думаю, уже знаете:

- t.me/zhikhkirill
- vk.com//zhikh.localhost
- github.com/zhikh23

Теоретические вопросы

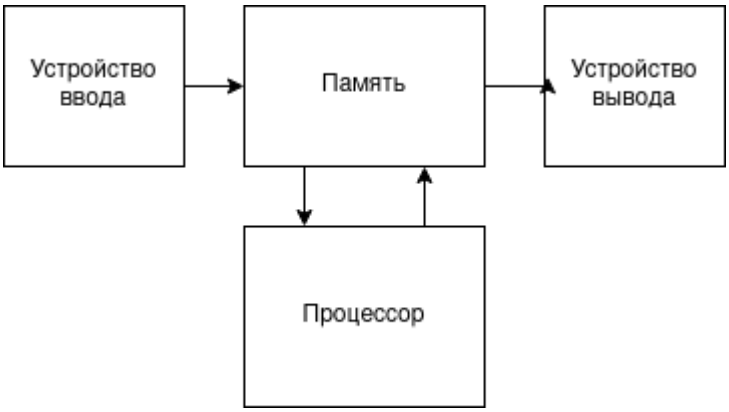
1. Электронная вычислительная машина. Устройство ЭВМ. Программа. Исходный текст, исполняемый файл

ЭВМ

ЭВМ -- основной вид реализации компьютеров, который технически выполнен на электронных элементах.

Компьютер -- устройство, способное выполнять заданную, чётко определённую, изменяемую последовательность операций (численные расчёты, преобразование данных и т. д.).

Устройство ЭВМ



> Примечание автора: сколько людей, столько и схем ЭВМ. На лекциях нам давали что-то похожее.

Программа и исходный текст

Исполняемая программа — сочетание компьютерных инструкций и данных, позволяющее аппаратному обеспечению вычислительной системы выполнять вычисления или функции управления.

Исходный текст программы — синтаксическая единица, которая соответствует правилам определённого языка программирования и состоит из инструкций и описания данных, необходимых для решения определённой задачи.

---

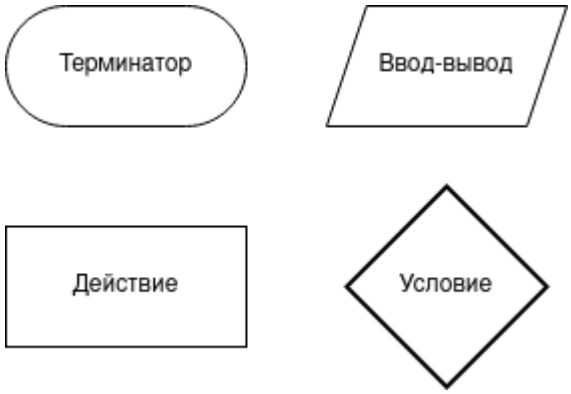
2. Схемы алгоритмов

Нестрогое определение

> От автора: PDF полной версии ГОСТа тут

Схема алгоритмов (она же *блок-схема*) -- схема, описывающая алгоритм или процесс в виде блоков различной формы, соединённых между собой линиями и стрелками.

Основные блоки



---

3. Языки программирования. Классификация

Язык программирования. Определение

Язык программирования -- формальный язык, предназначенный для записи компьютерных программ. Язык программирования определяет набор лексических, синтаксических и семантических правил, определяющих действия, которые выполнит ЭВМ под её управлением.

Классификация

- По уровню абстракции от аппаратной части:
  - низкоуровневые
  - высокоуровневые
- По способу выполнения исполняемой программы:
  - компилируемые
  - интерпретируемые
- По парадигме программирования:
  - императивные / процедурные языки
  - аппликативные / функциональные языки
  - языки системы правил / декларативные языки
  - объектно-ориентированные языки

---

4. Язык Python. Структура программы. Лексемы языка

Язык программирования Python

Python - высокоуровневый язык программирования общего назначения. Интерпретируемый. Является полностью объектно-ориентированным.

> Примечание автора: здесь и далее речь идёт о *третьей* версии -- `Python 3` (читается как *пайтон*). `Python 2` заметно отличается от последнего.

Структура программы

Программа -> Модули -> Операторы -> Выражения -> Объекты

Лексемы языка Python

Символы алфавита любого языка программирования образуют *лексемы*.

Лексема (token) – это минимальная единица языка, имеющая самостоятельный смысл. Лексемы формируют базовый словарь языка, понятный компилятору.

Всего существует пять видов лексем:

- ключевые слова (keywords)
  - Пример: `if`, `for`, `def` и т.п.
- идентификаторы (identifiers)
  - Пример: названия переменных, функций и т.п.
- литералы (literals)
  - Пример: `"hello world!"`, `42` и т.п.
- операции (operators)
  - Пример: `+`, `=`, `and`, `in` и т.п.
- знаки пунктуации (разделители, punctuators)
  - `,`, `;` и т.п.

---

5. Типы данных языка Python. Классификация. Скалярные типы данных. Приведение типов

Типы данных

Данные -- поддающееся многократной интерпретации представление информации в формализованном виде, пригодном для передачи, связи, или обработки.

Тип данных -- множество значений и операций над этими значениями.

Классификация

Основные способы классификации типов данных:

- скалярные и нескалярные;
- самостоятельные и зависимые (в том числе ссылочные).

Типы данных

Скалярные:

- Число -- `int`, `float`
- Логический тип -- `bool`

Нескалярные:

- Строка -- `str`
- Список -- `list`
- Словарь -- `dict`
- Кортеж -- `tuple`
- Множество -- `set`
- Файл

- Прочие основные типы
- Типы программных единиц
- Типы, связанные с реализацией

Приведение типов

Приведение типа -- преобразование значение одного типа в другое.

Бывает *явное* и *неявное*.

Неявное:

- 123 + 3.14

> Комментарий: здесь первое значение сначало *неявно* приводится к типу `float`, и лишь потом происходит сложение.

Явное:

- `int(3.14)`
- `str(obj)`

---

6. Операции над скалярными типами данных. Приоритеты операций

Над числами

- `x + y` -- сложение
  - `40 + 2 = 42`
- `x - y` -- вычитание
  - `16 - 2 = 14`
- `x * y` -- умножение
  - `16 * 2 = 32`
- `x / y` -- деление
  - `3 / 2 = 1.5`
- `x // y` -- целочисленное деление
  - `5 // 2 = 2`
- `x % y` -- остаток от деления
  - `5 % 2 = 1`
- `x ** y` -- возведение в степень
  - `2 ** 5 = 32`
- `-x` -- унарный минус
  - `x = 2; -x = -2`
- `+x` -- если бы мы знали, что это такое, но мы не знаем, что это такое
- `x | y` -- побитовое ИЛИ
  - `0b0101 | 0b0011 = 0b0111`
- `x & y` -- побитовое И
  - `0b0101 & 0b0011 = 0b0001`
- `x ^ y` -- побитовый ИСКЛЮЧАЮЩЕЕ ИЛИ
  - `0b0101 ^ 0b0011 = 0b0010`

- `~x` -- побитовое отрицание
  - `~0b0101 = 0b1010`
- `x << y` -- побитовый сдвиг влево
  - `0b11010110 << 2 = 0b01011000`
- `x >> y` -- побитовый сдвиг вправо
  - `0b11010110 >> 2 = 0b00110101`

Над логическими значениями

> Примечение автора: смотрим документацию и видим:
> *The `bool` class is a subclass of `int`*
> А значит для него определены *почти* (есть нюансы) все операции, что и для чисел. `True` эквивалетно `1`, а `False` -- `0`.

- `and` -- логическое И
- `or` -- логическое ИЛИ
- `not` -- логическое отрицание

Приоритеты операций

- `**`
- `~x`
- `+x`, `-x`
- `*`, `/`, `//`, `%`
- `+`, `-`
- `<<`, `>>`
- `&`
- `^`
- `|`
- `in`, `not in`, `is`, `is not`, `<`, `<=`, `>`, `>=`, `!=`, `==`
- `not x`
- `and`
- `or`

---

7. Функции ввода и вывода. Ввод данных

Ввод

```
# String
name = input("Enter your name: ")

# Integer
num = int(input("Enter integer number: "))

# Float
some_float_value = float(input())
```

```
# List
list_of_strings = input().split()
```

`prompt` -- текст-приглашение к вводу

Вывод

```
print(*objects, sep=" ", end="\n", file=sys.stdout, flush=True)
```

- `*objects` -- любое количество объектов, являющихся строками или поддерживающие приведение к ним (метод `__str__`).
- `sep` -- он же сепаратор. Строка, которая будет при выводе вставлена между отдельно переданными строками (см. пример ниже). По умолчанию -- пробел.
- `end` -- строка, которая будет добавлена в конец вывода. По умолчанию -- `\n`, т.е. перевод на новую строку.
- `file` -- куда будет напечатан результат. Обычно -- `sys.stdout`, `sys.stderr` или обыкновенный файл. По умолчанию -- `sys.stdout`.

```
# Examples
print("Hello, world!")

print("Hello", "world", sep=", ", end="!\n")    # Hello, world!

print("Error: something is wrong", file=sys.stderr)
```

---

8. Функции ввода и вывода. Функция вывода. Форматирование вывода

> Тоже самое, как и в вопросе 7

Форматирование вывода

```
replacement_field ::= "{" [field_name] ["!" conversion] [":" format_spec]
"}"
conversion ::= "r" | "s" | "a"
format_spec       ::= [[fill]align][sign][#][0][width][grouping_option]
["."precision][type]
fill              ::= any
align             ::= "<" | ">" | "=" | "^"
sign              ::= "+" | "-" | " "
width             ::= digit+
grouping_option   ::= "_" | ","
precision         ::= digit+
type              ::= "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" |
"n" | "o" | "s" | "x" | "X" | "%"
```

> От автора: "что ЭТО такое?!" -- это всего лишь простая форма Бэкуса — Наура (БНФ). Не волнуйтесь, её читать несложно. Особенно после регулярных выражений. Чтобы было ещё понятнее, покажу на примере.

Пример для `float`:

```
number = 42.0
print(f"{number:'='^12.5f}")  # out: '==42.00000=='
#        ^^^^^^  ^ ^ ^ ^^
# field_name _|  | | | ||
#          fill _| | | ||
#           align _| | ||
#             width _| ||
#            pecision _||
#                 type _|
```

- `field_name` -- что форматируем
- `fill` -- чем заполняем пустоты, которые образуются при выравнивании (`align`)
- `align` -- выравнивание `<`, `>`, `=`, `^`
- `width` -- ширина выравнивания
- `precision` -- точность указываемого значения для `float`
- `type` -- как форматировать

Пример для строк `str`:

```
s = "Hello!"
print(f"{s!r:-<12}")    # 'Hello!'----
#         ^
#         |_ conversion
```

- `conversion` -- как выводить строку (например, `r` экранирует все спец. символы и добавляет `'` на границах)

---

9. Оператор присваивания. Множественное присваивание

Оператор присваивания

Оператор присваивания предназначен для связывания имен со значениями и для изменения атрибутов или элементов изменяемых объектов. Оператор присваивания связывает переменную с объектом. Обозначается `=`.

Множественное присваивание

Примеры:

```
a, b = "foo", "bar"

a, b = b, a

pos = (0, 4)
x, y = pos
```

Комбинированное присваивание

- `+=`
- `-=`
- `*=`
- `/=`
- `//=`
- `%=`
- `**=`
- `&=`
- `|=`
- `>>=`
- `<<=`

Выполняет действие над значеним и присваивает результут тому же имени.

---

10. Условный оператор. Полные условные операторы. Неполные условные операторы. Тернарный оператор условия. Примеры использования

Полный условный оператор

```
if expr1:
    do_1()
elif expr2:
    do_2()
else:
    do_else()
```

Неполный условный оператор. Пример

```
max_value = 0
if x > max_value:
    max_value = x
```

Тернарный оператор

```
result = value_1 if condition else value_2
```

Эквивалетно

```
if condition:
    result = value_1
else:
    result = value2
```

Пример:

```
max_value = x if x > y else y
```

---

11. Условные операторы. Множественный выбор. Вложенные операторы условия. Примеры использования

Множественный выбор. Пример

Пример с некоторой реализацией меню:

```
cmd = input()
if not cmd:
    pass
elif cmd == "q":
    quit()
elif cmd == "m":
    menu()
elif cmd == "a":
    action()
else:
    print("Неизвестная команда")
```

Вложенные операторы условия. Пример

Пример с обработкой аргументов командной строки (почему бы и нет?)

```
arg = sys.argv[1]
if arg.startswith("--"):
    if arg == "--help":
        help()
    elif arg == "--interactive":
        run_interactive()
```

```
    elif arg == "--debug":
        debug()
    else:
        print(f"Неизвестный параметр:", arg)
        usage()
        exit(2)
elif arg.startswith("-"):
    if arg == "-h":
        help()
    elif arg == "-i":
        run_interactive()
    elif arg == "-d":
        debug()
    else:
        print(f"Неизвестный параметр:", arg)
        usage()
        exit(2)
```

---

12. Операторы цикла. Цикл с условием. Операторы break и continue. Примеры использования

Цикл. Определение

Цикл -- разновидность управляющей конструкции в высокоуровневых языках программирования, предназначенная для организации многократного исполнения набора инструкций.

Операторы цикла

- `while`
- `for`

Цикл с условием

```
while condition_is_true:
    do_something()
else:
    do_if_no_breaked()
```

Операторы break и continue

`break` -- переходит за пределы ближайшего заключающего цикла (после всего оператора цикла)

```
while y < size:
    while x < size:   # <---+
        if x == 5:    #     | continue
```

```
            continue  # >---+
        print(x, y)
```

`conrinue` -- переходит в начало ближайшего заключающего цикла (в строку заголовка цикла)

```
while y < size:  # <--------+
    while x < size:   #     |
        if x == 5:    #     | break
            break     # >---+
        print(x, y)
```

---

13. Операторы цикла. Цикл с итератором. Функция range(). Примеры использования

Цикл. Определение

Цикл -- разновидность управляющей конструкции в высокоуровневых языках программирования, предназначенная для организации многократного исполнения набора инструкций.

Операторы цикла

- `while`
- `for`

Цикл с итератором

```
for iterator in iterable:
    do_something()
else:
    do_if_no_breaked()
```

Функция range()

```
range(start = 0, stop, step = 1)
```

Порождает серию целых чисел `start <= n < stop` с шагом `step`.

> А вот тут в лекциях, очевидно, ошибка. Рабочий контр-пример ниже. Поэтому приведу своё определение
> Функция `range(start, stop, step)` возвращает объект, создающий последовательность чисел, начинающуюся с `start`, изменяемая каждую итерацию на `step` и останавливающуюся, когда достигает значения `stop`.

```
for i in range(5, -1, -1):
    print(i)
# 5, 4, 3, 2, 1, 0
```

---

14. Изменяемые и неизменяемые типы данных

Неизменяемые:

- `int`
- `float`
- `str`
- `bytes`
- `tuple`

Изменяемые

- `list`
- `dict`
- `set`
- и др.

Неизменяемые типы данных, как ни странно, не изменяемы: (`id` -- возвращает уникальный идентификатор объекта)

```
>>> a = 5
>>> id(a)
139709610098536
>>> a += 1
>>> id(a)
139709610098568
```

При попытке изменить неизменяемое значение, имени присваивается другой участок памяти (см. пример выше).

Изменяемые же ведут себя предсказуемо:

```
>>> l = [1, 2]
>>> id(l)
139709375880640
>>> l.append(3)
>>> id(l)
139709375880640
```

Из этого следует поведение неизменяемых и изменяемых объектов при передачу в функцию:

```python
def inc(x: int):
    x += 1

a = 5
inc(a)
print(a)    # 5


def append_one(x: list):
    x.append(1)

l = [1, 2]
append_one(l)
print(l)    # [1, 2, 1]
```

---

15. Списки. Основные функции, методы, операторы для работы со списками

---

16. Списки. Создание списков. Списковые включения

---

17. Списки. Основные методы для работы с элементами списка. Добавление элемента, вставка, удаление, поиск

---

18. Списки. Основные операции со списками. Поиск минимального элемента. Поиск максимального элемента. Нахождение количества элементов. Нахождение суммы и произведения элементов

---

19. Списки. Использование срезов при обработке списков

---

20. Кортежи. Основные функции, методы, операторы для работы с кортежами

---

21. Словари. Понятие ключей и значений. Создание словарей. Основные функции, методы, операторы для работы со словарями

---

22. Множества. Основные функции, методы, операторы для работы с множествами

---

23. Строки. Основные функции, методы, операторы для работы со строками. Срезы

---

24. Матрицы. Создание матрицы. Ввод и вывод матрицы. Выполнение операций с элементами матрицы

---

25. Матрицы. Квадратные матрицы. Обработка верхне- и нижнетреугольных матриц. Работа с диагональными элементами матрицы

---

26. Отладка программы. Способы отладки

---

27. Подпрограммы. Функции. Создание функции. Аргументы функции. Возвращаемое значение

---

28. Функции. Области видимости.29. Функции. Завершение работы функции. Рекурсивные функции. Прямая и косвенная рекурсия

---

30. Функции высшего порядка. Замыкания

---

31. lambda-функции

---

32. Аннотации

---

33. Функции map, filter, reduce, zip

---

34. Декораторы

---

35. Знак “_“. Варианты использования

---

36. Модули. Способы подключения

---

37. Модуль math. Основные функции модуля. Примеры использования функций

---

38. Модуль time

---

39. Модуль random. Работа со случайными числами

---

40. Модуль copy. Способы копирования объектов различных типов. “Глубокая” и “мелкая” копии

41. Объектно-ориентированное программирование. Основные понятия ООП

42. Исключения

43. Файлы. Программная обработка файлов. Понятие дескриптора. Виды файлов

44. Файлы. Режимы доступа к файлам

45. Файлы. Текстовые файлы. Основные методы для работы

46. Файлы. Текстовые файлы. Чтение файла. Запись в файл. Поиск в файле

47. Файлы. Текстовые файлы. Итерационное чтение содержимого файла

48. Файлы. Бинарные файлы. Основные методы. Сериализация данных

49. Файлы. Оператор with. Исключения при работе с файлами

50. Типы данных bytes и bytearray. Байтовые строки. Конвертация различных типов в байтовые строки и обратно

51. Модуль struct

52. Модуль os. Основные функции

53. Генераторы. *

54. Модуль numpy. Обработка массивов с использованием данного модуля. Работа с числами и вычислениями

55. Модуль matplotlib. Построение графиков в декартовой системе координат. Управление областью рисования

56. Модуль matplotlib. Построение гистограмм и круговых диаграмм

57. Списки. Сортировка. Сортировка вставками. Сортировка выбором

---

58. Списки. Сортировка вставками. Метод простых вставок. Метод вставок с бинарным поиском. Вставки с барьером. Метод Шелла

---

59. Списки. Сортировка. Обменные методы сортировки. Сортировка пузырьком. Сортировка пузырьком с флагом. Метод шейкер-сортировки

---

60. Списки. Сортировка. Метод быстрой сортировки