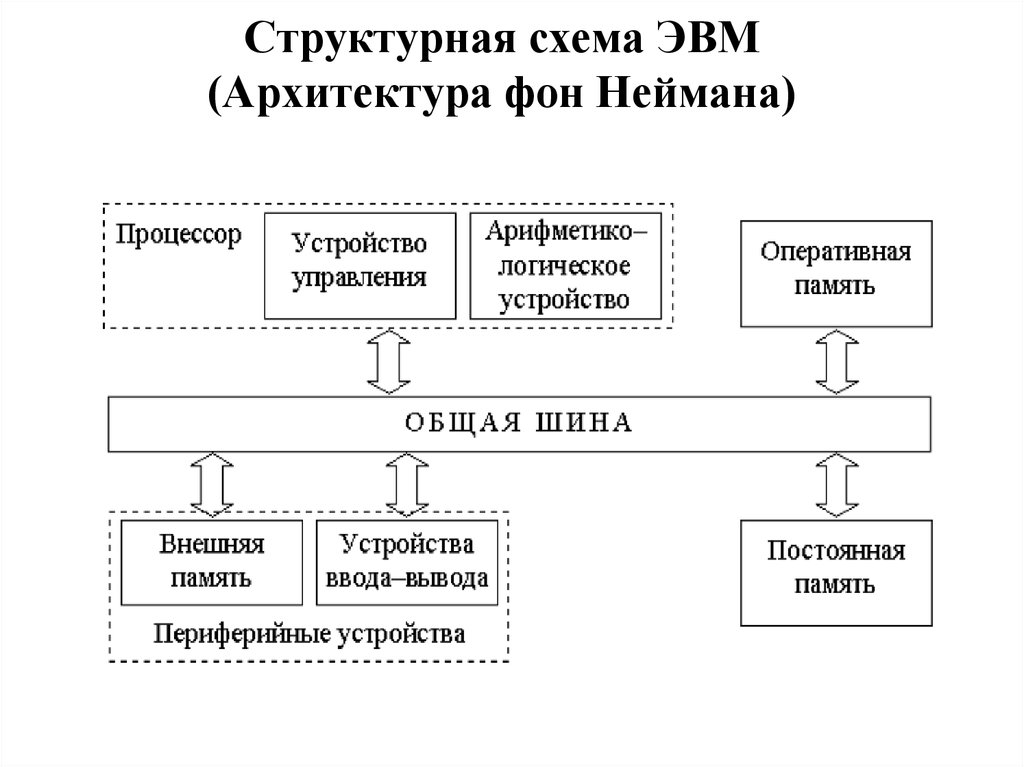
Вопросы для подготовки к экзамену по дисциплине “Программрование”

Электронно-вычислительная машина. Устройство ЭВМ.

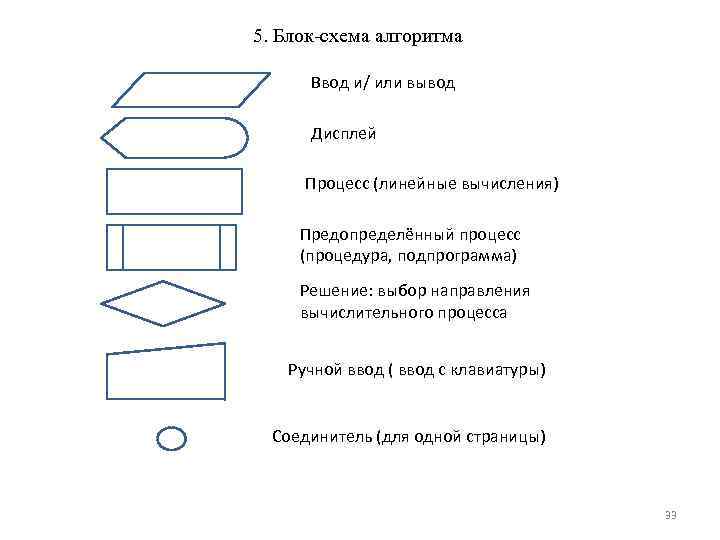
* Компьютер – это устройство, способное выполнять заданную, чётко определённую, изменяемую последовательность операций (численные расчёты, преобразования данных и тд)
* Электронно-вычислительная машина (сокращённо ЭВМ) — комплекс технических, аппаратных и программных средств, предназначенных для автоматической обработки информации, вычислений, автоматического управления. Является основным видом реализации компьютеров, который технически выполнен на электонных элементах.
* Устройство ЭВМ

1. Процессор – интегральная схема, исполняющая машинные инструкции (код программ) главная часть аппаратного обеспечения компьютера.
2. Системная шина – соединение, служащее для передачи данных между функциональными блоками компьютера.
3. ПЗУ – быстрая, энергонезависимая память, которая, предназначенная только для чтения. Информация заносится в нее один раз (обычно в заводских условиях) и сохраняется постоянно (при включенном и выключенном компьютере).
4. ОЗУ – представляет собой область временного хранения данных, при помощи которой обеспечивается функционирование программного обеспечения.
5. Накопитель на жёстком магнитном диске – [запоминающее устройство](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BF%D0%BE%D0%BC%D0%B8%D0%BD%D0%B0%D1%8E%D1%89%D0%B5%D0%B5_%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE) (устройство хранения информации, [накопитель](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%BA%D0%BE%D0%BF%D0%B8%D1%82%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) [произвольного доступа](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B8%D0%B7%D0%B2%D0%BE%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%B4%D0%BE%D1%81%D1%82%D1%83%D0%BF), основанное на принципе магнитной записи. Является основным [накопителем данных в большинстве компьютеров](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C).
6. Накопитель на оптическом диске – собирательное название для [носителей информации](https://ru.wikipedia.org/wiki/%D0%9D%D0%BE%D1%81%D0%B8%D1%82%D0%B5%D0%BB%D1%8C_%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%B8), выполненных в виде дисков, чтение с которых ведётся с помощью [оптического](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D1%82%D0%B8%D0%BA%D0%B0) ([лазерного](https://ru.wikipedia.org/wiki/%D0%9B%D0%B0%D0%B7%D0%B5%D1%80%D0%BD%D0%BE%D0%B5_%D0%B8%D0%B7%D0%BB%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)) излучения.
7. АЛУ – блок процессора, который служит для выполнения арифметических и логических преобразований над данными.
8. МПП – предназначена для кратковременного хранения, записи и выдачи информации.
9. УУ - формирует и подает во все блоки машины управляющие импульсы; выдает адреса требуемых ячеек памяти, и передает их в другие блоки ЭВМ.

Программа. Исходный текст, исполняемый файл.

* Программа – это описанная на формальном языке последовательность действий, которые необходимо выполнить над данными для решения поставленной задачи.
* Исходный текст программы – синтаксическая единица, которая соответствует правилам определённого языка программирования, состоящая из определений и операторов или инструкций, необходимых для решения определённой задачи.
* Исполняемый файл – файл, содержащий программу в виде, в котором она может быть исполнена компьютером (то есть в машинном коде).

Схемы алгоритмов.

* Схема — это абстракция какого-либо процесса, наглядно отображающая наиболее значимые части.
* Блок-схема — распространённый тип [схем](https://ru.wikipedia.org/wiki/%D0%A1%D1%85%D0%B5%D0%BC%D0%B0_(%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D0%BA%D0%B0)), описывающих [алгоритмы](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) или процессы, в которых отдельные шаги изображаются в виде блоков различной формы, соединённых между собой линиями, указывающими направление последовательности.
* Данные, носитель которых неопределён (ввод / вывод)
* Ввод вручную
* Отображающее устройство (вывод данных на экран)
* Процесс
* Решение (проверка условий)
* Терминатор (начало или конец программы)
* Соединитель – переход в другую часть схемы
* Границы цикла
* Линии

Языки программирования. Классификация.

* 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 Ассемблер, машинные коды
* Высокоуровневые (Python, C, C++, Pascal, Java)
* По способу выполнения исполняемой программы:
* Компилируемые (C, C++, Pascal)
* Интерпретируемые (Python, Java)
* По парадигме программирования:
* Императивные / процедурные Pascal, C
* Аппликативные / функциональные (Hascell, Lisp)
* Языки системы правил / декларативные (Prolog)
* Объектно-ориентированные Java, C++, Python

Язык Python. Структура программы. Лексемы языка.

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

Программа

Модули

Операторы

Выражения

Объекты

* Лексема (token) – это минимальная единица языка, имеющая самостоятельный смысл. Лексемы формируют базовый словарь языка, понятный компилятору.
* Виды лексем:
* Ключевые слова (keywords) (True, False, if, else, for, in, and, or, not, None)
* Идентификаторы (identifiers) (a, matrix, arr)
* Литералы (literals) (123; 1.7; ‘abc’; [1, 2, 3])
* Операции (operators) (<, >, =, +, -, \*)
* Знаки пунктуации (punctuators)

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

* [Тип данных](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) (англ. Data type) - характеристика, определяющая множество допустимых значений и набор операций, которые можно осуществлять на этих значениях.
* Числа
* Строки
* Списки
* Словари
* Кортежи
* Файлы
* Множества
* Прочие основные типы
* Типы программных единиц
* Типы, связанные с реализацией
* Классификация
* Простые и сложные (массивы, файлы, строки, множества, словари и тд)
* Скалярные и нескалярные (агрегатные)
* Самостоятельные и зависимые (ссылки)
* Скалярные типы данных – поддерживающий арифметические операции (в том числе деление, возведение в степень и тд)
* Целые
* Вещественные
* Логический
* Преведение
* Неявное (123 + 3.14)
* Явное (int(x, base = 10); complex([real[, imag]])

Операции над скалярными типами данных. Приоритеты операций.

* Приоритетность:

1. Возведение в степень
2. Умножение, деление, взятие остатка
3. Сложение, вычитание
4. Побитовое И (&)
5. Побитовое исключающее ИЛИ (^)
6. Побитовое ИЛИ (|)
7. Сдвиги (>>, <<)

Функции ввода и вывода.

* input([prompt])
* print(\*objects, sep=‘’, end=‘\n’, file=sys.stdout, flush=False)

Функция вывода. Форматирование вывода.

* print(\*objects, sep=‘’, end=‘\n’, file=sys.stdout, flush=False)
* Форматирование текста (функция format())

[[fill]align][sign][#][0][width][grouping\_option][.precision][type]

fill ::= any character

align ::= "<" | ">" | "=" | "^"

sign ::= "+" | "-" | " "

width ::= integer

gruping\_option ::= ‘\_’ | ‘,’

precision ::= integer

type ::= "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" | "n" | "o" | "s" | "x" | "X" | ''%''

**print**(format(-123456789, '?^+40\_.2f'))

* Форматирование строк

str.format(\*args, \*\*kwargs)

replacement\_field ::= "{" [field\_name] ["!" conversion] [":" format\_spec] "}"

field\_name ::= arg\_name ("." attribute\_name | "[" element\_index "]")\*

arg\_name ::= [identifier | digit+]

attribute\_name ::= identifier

element\_index ::= digit+ | index\_string

index\_string ::= <any source character except "]"> +

conversion ::= "r" | "s" | "a"

format\_spec ::= <аналогично функции format()>

**print**('{:.4f}'.format(123,7))

Оператор присваивания. Множественное присваивание.

* Оператор присваивания предназначен для связывания имен со значениями и для изменения атрибутов или элементов изменяемых объектов.
* =
* name = value
* Позиционное присваивание

name1, name2, …, nameN = value1, value2, …, valueN

* Комбинированное присваивание

+=

-=

\*=

/=

//=

%=

\*\*=

&=

|=

>>=

<<=

* Множественное присваивание позволяет присвоить нескольким объектам одно значение

var1 = var2 = var3 = value

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

* Условный оператор выбирает, какое действие следует выполнить, в зависимости от значения переменных в момент проверки условия.

**if** expression1:

block1

**elif** expression2:

block2

**else**:

block3

**if** x > y: m = x

**else**: m = y

* Тернарный оператор условия:

x **if** c **else** y

(самый низкий приоритет среди всех операций языка)

* Примеры:

1) Сравнить 2 числа:

**if** x > y:

print('x больше y')

**elif** x < y:

print('x меньше y')

**else**:

print('Числа равны’)

2) Найти максимальное из 2х чисел

m = x

**if** y > x:

m = y

3) Найти максимальное из 2х чисел

m = x **if** x > y **else** y

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

* Множественный выбор:

**if** expression1:

block1

**elif** expression2:

block2

**else**:

block3

* Вложенные операторы условия:

**if** expression1:

**if** expression2:

block1

**else**:

block2

**else**:

block3

* Примеры использования:

1) Нахождение максимального из 3х чисел

**if** x > 0:

**if** x > 10:

print('Число больше 10.')

**else**:

print('Число положителшьное, но меньше 10.')

**else**:

print('Число отрицаьельное.')

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

* while, for

Циклы предназначены для организации многократного исполнения инструкций.

* Цикл while используется в Python для неоднократного исполнения определенной инструкции до тех пор, пока заданное условие остается истинным.

**while** condition1:

block1

**else**:

block2

* break – переходит за пределы ближайшего заключающего цикла (после всего оператора цикла)
* continue – переходит в начало ближайшего заключающего цикла (в строку заголовка цикла)
* Примеры использования:

1) Удалить из списка первый нулевой элемент

i = 0

**while** i < len(arr):

**if** arr[i] == 0:

arr.pop(i)

**break**

i += 1

**else**:

print('В списке нет нулевых элементов.')

2) Вывести все отрицательные элементы списка

i = 0

**while** i < len(arr):

**if** arr[i] > 0:

**continue**

print(arr[i])

i += 1

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

* Цикл с итератором – средство для перебора итерируемых объектов:

**for** var **in** object:

block1

**else**:

block2

* range(start=0, stop, step=1) – порождает серию целых чисел start <= n < stop с шагом step
* Примеры использования:

1) Найти произведение элементов списка

p = 1

**for** i **in** range(len(arr)):

p \*= arr[i]

Изменяемые и неизменяемые типы данных.

* Неизменяемые (immutble)
* bool
* int
* float
* tuple
* str
* frozenset
* range
* Изменяемые (mutable):
* list
* set
* dict

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

* Списки в Python - упорядоченные изменяемые коллекции объектов произвольных типов.
* Функции для работы со списками:
* all() – возвращает True, если все элементы истинны или список пуст
* enumerate(iterable, start=0) – возвращает перечисляемый объект – кортежи (индекс значение)
* len(s) – количество элементов
* list(iterable) – создание списка на основе итерируемого объекта
* max(iterable)
* min(iterable)
* reversed(seq) – возвращает итератор. Не создаёт копию последовательности.
* sorted(iterable, key=None, reverse=False) – в параметре key указывается функция, которая будет сортировать
* sum(iterable)
* Методы для работы со списками:
* append(x)
* extend(utterable) – расширение списка с помощью итерируемого объекта
* insert(i, x) – вставка элемента в i-тую позицию, если i за границами списка, то вставка в конец или начало.
* remove(x) – удаление первого элемента по значению
* pop(i) – удаляет элемент в i-той позиции
* clear()
* index(x[, start[, end]]) – возвращает индекс первого элемента, равного x (можно задать диапазон)
* count(x)
* sort(key = None, reverse = False)
* reverse() – разворачивает элементы последовательности
* copy()
* Операторы:
* +
* \*
* in
* del (удаление переменной или элемента)
* ==
* <, >, <=, >=
* for i in list:
* Срезы – извлечение из списка некоторого фрагмента:
* [start:stop:step]
* a[:] – все элементы списка
* a[5:] - начиная с 5 включительно
* a[:2] – до 2 не включительно
* a[::-1] – разворачивание списка

Списки. Создание списков. Списковые включения.

* Создание списков:
* a = []
* a = list()
* a = [0] \* 10
* a = [i \* i **for** i **in** range(10)]
* a = [i **for** i **in** range(10) **if** i % 2 == 0]
* Списковые включения можно использовать для создания списков из других итерируемых объектов, применяя выражение или функцию к каждому элементу.

[x + 1 **for** x **in** (1, 2, 3)]

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

* Методы для работы со списками:
* append(x)
* extend(utterable) – расширение списка с помощью итерируемого объекта
* insert(i, x) – вставка элемента в i-тую позицию, если i за границами списка, то вставка в конец или начало.
* remove(x) – удаление первого элемента по значению
* pop(i) – удаляет элемент в i-той позиции
* clear()
* index(x[, start[, end]]) – возвращает индекс первого элемента, равного x (можно задать диапазон)
* count(x)
* sort(key = None, reverse = False)
* reverse()
* copy()
* Вставка элемента:

num = int(input('Введите элемент, который хотите вставить: '))

arr.append('\*')

**for** i **in** range(len(arr) - 1, num, -1):

arr[i] = arr[i - 1]

arr[num] = int(input('Введите число, которое хотите вставить: '))

* Удаление элемента:

num = int(input('Введите номер удаляемого элемента: '))

**for** i **in** range(num, len(arr) - 1:

arr[i] = arr[i + 1]

arr.pop(-1)

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

* min()
* max()
* count()
* sum()

**import** functools

mas = [1, 2, 3]

**print**(functools.reduce(**lambda** x, y: x \* y, mas))

Списки. Использование срезов при обработке списков.

* [start:stop:step]
* a[:] – все элементы списка
* a[5:] - начиная с 5 включительно
* a[:2] – до 2 не включительно
* a[::-1] – разворачивание списка

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

* Кортежи – неизменяемая последовательность.
* Создание:
* k = tuple()
* k = (1, 2, 3)
* k = 1, 2, 3
* Функции:
* sorted()
* len()
* sum()
* reversed()
* all()
* enumerate()
* min()
* max()
* Методы:
* index()
* count()
* Операторы:
* in
* not in
* +
* \*

Словари. Понятие ключей и значений. Создание словарей.

* Словари – это неупорядоченные коллекции произвольных объектов с доступом по ключу. Аналоги в других языках – ассоциативные массивы, хэш-таблицы.
* Создание:
* a, b = dict(), {'odd': 1, 'even':2}
* c = dict(short='dict', long='dictionary')
* b = dict([(1, 1), (2, 2)])
* e = {a:a\*\*2 **for** a **in** range(7)}
* f = dict.fromkeys(['a', 'b'], 100)
* Ключи должны быть хэшируемыми (\_\_hash\_\_()) и сравнимыми ( \_\_eq\_\_())
* Уникальными, не изменяемыми
* Ключ – илентификатор элемента в словаре.
* Значение – элемент, соответствующий ключу

Основные функции, методы, операторы для работы со словарями.

* Функции:
* len()
* str()
* type()
* dict()
* sum() – применимо к клчам
* min() – применимо к ключам
* max() – применимо к ключам
* Операторы
* del, in, not in, |
* Методы:
* clear()
* copy()
* fromkeys(iterable[, value]) – создаёт словарь на основе ключей и значений по умолчанию
* get(key[, default]) – возвращает значение по ключу (если ключ не найден, то вернёт None)
* items() – возвращает отображение содержимого
* keys() – возвращает отображение ключей
* pop(key[, default]) – удаляет значение и возвращает его
* popitem() – возвращает последнюю добавленную в словарь пару или порождает исключение
* setdefault(key[, default]) – значение по умолчанию для метода get на случай отсутствия ключа
* update([other]) – обновляет значения по другому словарю (кортежу)
* values() – возвращает отображение значений

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

* Множества — это неупорядоченная коллекция уникальных элементов.
* Элементы должны быть хэшируемыми
* Создание:

a, b = set(), {1, 2, 3}

* Функции:
* len()
* sum()
* min()
* max()
* Методы:
* isdisjoint(other) – нет пересечения
* issubset(other) – является подмножеством
* issuperset(other) – является надмножеством
* union(\*others) – объединение множеств
* intersection(\*others) – пересечение множеств
* difference(\*others) – различные элементы
* symmetric\_difference(other) – элементы из множеств, исключая общие
* copy()
* update(\*others) – добавление нескольких элементов в множество
* intersection\_update(\*others)
* difference\_update(\*others)
* symmetric\_difference\_update(other)
* add(elem)
* remove(elem)
* discard(elem) (удаляет без ошибки, если нет элемента)
* pop()
* clear()
* Операторы:
* in
* not in
* >=, <=, >, <
* |, &, -, ^

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

* Строка – тип данных, значениями которога является произвольная последовательность символов. Обычно реализуется как массив символов.
* Str(object, encoding = ‘utf-8’, errors=‘strict’)
* Неизменяемый тип данных
* Функции:
* chr() – число в символ таблицы ASCI
* ord() – символ в число
* len()
* str()
* max() – элемент с максимаольным значением по ascii
* min() – элемент сминимальным значением по ascii
* Методы:
* count(sub[, start[, end]])
* center(width[, fillchar])
* encode() – строку в виде байтового объекта возвращает
* find(sub[, start[, end]]) – индекс первого совпадения (если совпадения нет, то -1)
* format(\*args, \*\*kwargs)
* index(sub[, start[, end]]) – если заданная строка не найдена, то порождает исключение
* isallnum() – все символы являются числами или буквами
* isalpha() – все символы буквы
* isdigit()  – все символы числовые
* islower()
* isupper()
* join() – возвращает конкатенацию строк в итерируемом объекте
* lower() – перевод символов строки в нижний регистр
* upper() – перевод символов строки в верхний регистр
* strip() – копия строки с удалением крайних пробелов
* replace(old, new[, count])
* split(sep=None) – список строк
* splitlines()
* Операторы:
* +
* \*
* in
* >=, <=, >, <

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

* В математике – таблица чисел
* В программировании – массив массивов (двумерный массив)
* Создание матрицы:

a = [[1, 2, 3], [3, 2, 1]]

a = [[0] \* m **for** i **in** range(n)]

* Ввод / вывод:

n = int(input())

m = int(input())

a = []

**for** i **in** range(n):

a.append([])

**for** j **in** range(m):

a[i].append(int(input()))

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

* Квадратная матрица – у которой количество строк и столбцов одинаковое
* Верхнетреугольные:

**for** i **in** range(n):

**for** j **in** range(i, n):

* Нижнетреугольные:

**for** i **in** range(n):

**for** j **in** range(i + 1):

* Главная диагональ:

**for** i **in** range(n):

a[i][i]

* Побочная диагональ:

**for** i **in** range(n):

a[i][n - 1 - i]

Отладка программы. Способы отладки.

* Отладка – этап разработки компьютерной программы, на котором обнаруживают, локализируют и устраняют ошибки. При отладке требуется:
* Узнавать текущие значения переменных
* Выяснять, по какому пути выполнялась программ
* Способы отладки:
* Использование отладочной печати
* Использование отладчика

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

* Подпрограмма – поименованная или иным образом идентифицированная отдельная функционально независимая часть компьютерной программы.
* Процедуры
* Функции
* Функция в Python — это фрагмент кода для повторного использования, который применяется для выполнения одного связанного действия. Функция принимает на вход некоторые параметры и возвращает значения.
* Оператор def создаёт новый объект и присваивает его имени

def имя\_функции(список формальных параметров):

операторы

* Аргументы функции – переменные, которые вызывающая программа даёт функции
* Формальные аргументы – те, которые объявлены при описании функции
* Фактические аргументы – те, которые передаются в функцию при её вызове.

* Модификация аргументов внутри функции:
* Неизменяемого – создаёт копию (не повлияет на вызывающий код)
* Изменяемого – повлияет на вызывающий код (изменит значение в нем)
* Виды параметров в Python:
* Позиционные аргументы (определяются по их позиции при объявлении функции)

**def** f(a, b, c):

...

f(c, a, b)

* Именованные аргументы

**def** f(a=3, b=2, c=1):

...

f(a=1, c=2)

* Возвращаемое значение передаётся из функции в то место, откуда она была вызвана с помощью return. Функция может и не возвращать значения.

Функции. Области видимости.

* Область видимости – часть программы, в пределах которой идентификатор остаётся связан с сущностью, которой он был назначен при объявлении.
* Имена, присвоенные внутри def, видны только в коде внутри этого оператора, ссылаться на них извне функции нельзя.
* Имена внутри def не конфликтуют с переменными за пределами def.
* Области видимости:
* Глобальная – если переменная объявлена за пределами всех def, то она является глобальной в целом файле.
* Локальная – переменная, объявленая внутри def, будет локальной в своей функции.
* Нелокальная– переменная, объявленная внутри def, включающем другие def.
* Встроенная (built-in)

* Оператор global делает имя внутри функции глобальным
* Оператор nonlocal делает имя внутри функции нелокальным

Функции. Завершение работы функции. Рекурсивные функции. Прямая и косвенная рекурсия.

* Завершение работы:
* Функция выполнила все действия
* Функция дошла до оператора return
* Рекурсивные функции – вызывающие сами себя:
* Непосредственно – прямая рекурсия

**def** factorial(n):

**if** n == 1:

**return** n

**else**:

**return** n\*factorial(n-1)

* Через другие подпрограммы – косвенная рекурсия

**def** f(n):

**if** n == 1:

**return** 1

**return** 3 \* g(n - 1)

**def** g(n):

**if** n == 1:

**return** 1

**return** f(n - 1)

print(f(77))

* Тело рекурсивной подпрограммы должно иметь не меньше двух альтернативных (условных) ветвей, хотя бы одна из которых должна быть терминальной.
* Виды рекурсивных вызовов:

|  |
| --- |
| **def** outer():  x = 1  **def** inner():  nonlocal x  **print**(x)  **return** inner |

1. По количеству вызовов:

* Линейная (один вызов самой себя)
* Нелинейная (несколько вызовов самой себя)

1. По месту расположения рекурсивного вызова:

* Головная (рекурсивный вызов ближе к началу)
* Хвостовая (Рекурсивный вызов является последним оператором)

Функции высшего порядка. Замыкания.

* Фукция высшего порядка – та, которая принимает в качестве аргументов или возвращает другие функции
* Замыкания (closure)  в программировании – это функция первого класса, в теле которой присутствуют ссылки на переменные, объявленные вне тела этой функции в окружающем коде и не являющиеся её параметрами

lambda-функции.

* Оператор lambda создаёт и возвращает объект функции, который будет вызываться позднее, не присваивая ему имени
* Анонимная одноразовая функция.
* lambda аргумент1, аргумент2 …: выражение, использующее аргументы

double = **lambda** x: x \* 2

**print**(double(5))

* Используется для сокращения кода в тех местах, где включение оператора def не разрешено синтаксисом (например, с функциями map, filter, reduce).

|  |
| --- |
| **def** fibonachi():  x1 = 0  x2 = 1  **def** summa():  nonlocal x1, x2  x3 = x1 + x2  x1, x2 = x2, x3  **return** x3  **return** summa  f = fibonachi()  **for** i **in** range(2, 10):  **print**(f()) |

Аннотации.

* Аннотации – способ добавлять произвольные метаданные к аргументам функции и возвращаемому значению

**def** div(a: 'the divident',

b: 'the divisor') -> 'the result of dividing a by b':

**return** a / b

**def** sq\_sum(v1:int, v2:int) -> int:

**return** v1\*\*2 + v2\*\*2

Функции map, filter, reduce, zip.

* map(function, iterable)
* Возвращает итератор, применяющий функцию к каждому элементу итерируемого объекта

new\_list = list(map(int, old\_list))

* filter(function, iterable[, initializer])
* Возвращает итератор, фильтрующий объект по истинности или ложности переданной функции при текущем значении.

zolushka = list(filter(**lambda** x: x == 'мак', mixed))

* zip(\*iterable, strict=False)
* Соединяет элементы итерируемых объектов в кортежи
* Параметр strict приводит к исключению, если длина объектов отличается

a = [1,2,3]

b = "xyz"

c = (None, True)

res = list(zip(a, b, c))

**print** (res)

[(1, 'x', None), (2, 'y', True)]

* reduce(function, iterable[, initializer])
* Применяет функцию к элементам итерируемого объекта комулятивно (накопительно). Сначала к первым 2м элементам (либо отдельно заданному значению и первому элементу), далее к промежуточному результату и очередному значению.

**from** functools **import** reduce

p = reduce(**lambda** x, y: x \* y, items)

Декораторы.

* Декоратор – это функция, которая позволяет обернуть другую функцию для расширения её функциональности без непосредственного изменения её кода.

|  |
| --- |
| **def** bread(f):  **def** wrapper():  **print**('----------')  f()  **print**('----------')  **return** wrapper  **def** ingredients(f):  **def** wrapper():  **print**('Salad')  f()  **print**('Tomato')  **return** wrapper  @bread  @ingredients  **def** sendwich():  **print**('Bacon')  sendwich() |

@decorator\_function

**def** some\_decorated\_function():

Знак \_

* Хранение значения последнего выражения в интерпретаторе

>>> 10

10

>>> \_

10

>>> \_ \* 10

100

* Игнорирование некоторых значений (при разыменовании кортежей)

x, \_, y = (1, 2, 3) *# x = 1, y = 3*

x, \*\_, y = (1, 2, 3, 4, 5, 6, 7) *# x = 1, y = 7*

**for** \_ **in** range(10):

do\_something

**for** \_, val **in** enumerate(list):

do\_something

* Задание специальных значений для имен переменных или функций (\_name, \_\_name, \_\_name\_\_)

Модули. Способы подключения

* Модуль — это файл, содержащий код python, который можно импортировать и многократно использовать в других программах.
* Способы подключения:
* **import** module
* **import** module **as** m
* **import** module1 **as** m1, module2 **as** m2
* **from** module **import** a, b
* **from** module **import** \*

Модуль math. Основные функции модуля. Примеры использования функций.

* Этот модуль предоставляет обширный функционал для работы с числами.
* ceil(x) – округление вверх
* comb(n, k) – сочетания из n по k
* factorial(x) – факториал
* floor(x) – округление вниз
* isqrt(n) – целочисленный квадратный корень
* ldexp(x, i) – x \* (2 \*\* i)
* modf(x) – возвращает дробную и целую часть числа
* exp(x) – e^x
* log(x[, base]) – логарифм
* log2(x) – по основанию 2
* log10(x) – по основанию 10
* pow(x, y) – x^y
* sqrt(x) – корень
* acos(x)
* asin(x)
* atan(x)
* cos(x)
* sin(x)
* tan(x)
* degrees(x) – конвертирует радианы в градусы
* radians(x) – конвертирует градусы в радианы
* pi
* e
* inf
* nan

Модуль time.

* Time - модуль для работы со временем в Python.
* sleep(secs) – задержка в секундах
* time() – время выраженное в секундах с начала эпохи Unix, время с 01.01.1970 00:00
* clock() - в Unix, возвращает текущее время. В Windows, возвращает время, прошедшее с момента первого вызова данной функции.

Модуль random. Работа со случайными числами.

* Реализует генерацию псевдослучайных чисел различных распределений.
* Функции состояния:
* seed(a=None, version=2) – настраивает генератор на новую последовательность
* getstate() – внутреннее состояние генератора
* setstate(state) – восстанавливает внутреннее состояние генератора
* Функции генерации последовательности байтов:
* randbytes(n)
* Числовые функции:
* randrange([start], stop, [step])
* randint(a, b)
* getrandbits()
* Функции последовательностей:
* choice(seq) – случайный элемент последовательности
* choices(population, k=1) – случайные элементы (элементы могут повторяться)
* shuffle(x[, random]) – перемешивание элементов
* sample(population, k) – выбор элементов из набора (не повторяются)

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

* copy(x) – мелкая копия (копия объекта, если в объекте присутствуют вложенные элементы, то копируются только ссылки на них. Поэтому, если мы внесем какие-либо изменения во вложенный элемент объекта, это отразится на исходном объекте.)
* deepcopy(x, [memo]) – глубокая копия (рекурсивно вставляет в него копии объектов, находящихся в оригинале. Поэтому вложенные элементы так же будут копиями, а не ссылками.)

Исключения.

* Исключения – это тип данных, позволяющий классифицировать ошибки и обрабатывать их

**try**:

...

**except** ExceptionType [**as** err]:

processing code

**except** ExceptionType2:

processing code

**except** Exception:

processing code

**else**:

no errors

**finally**:

completion of processing

* Вызов исключения:

**raise** Exception('some error')

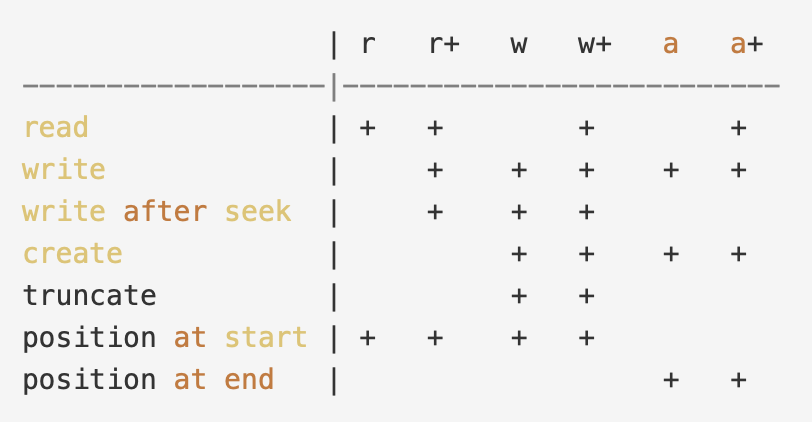
**raise** ValueError

* Цепочки исключений – в процессе одного исключения может произойти другое исключение, поэтому исключения будут обрабатываться по цепочке.
* Кроме того можно создавать свои классы исключений.

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

* Файл – поименованное место на диске (внешняя память).
* В языках программирования обычно применяется концепция, в которой файл является абстракцией, не привязанной к конкретному типу носителя и файловой системе, а работа с файлами осуществляется подобно обработке массива данных.
* Файловый дескриптор – целое число, которое присваивается операционной системой каждому потоку ввода-вывода при его создании
* Виды файлов:
* Текстовые
* Структурированные (типизированные) форматы
* Бинарные
* Формат файла определяется его содержимым. Расширение файла обычно соответствует формату файла, но в общем случае никак на него не влияет

Файлы. Режимы доступа к файлам.

* r - открывает файл только для чтения,
* w - открыт для записи (перед записью файл будет очищен),
* x - эксклюзивное создание, бросается исключение FileExistsError, если файл уже существует.
* a - открыт для добавления в конец файла
* + - символ обновления (чтение + запись).
* t - символ текстового режима.
* b - символ двоичного режима.

Файлы. Текстовые файлы. Основные методы для работы.

* Те́кстовый файл — компьютерный файл, содержащий текстовые данные.
* f.close – закрывает
* f.read() – читает весь файл или до введённого значения
* f.readline() - читает строку
* f.readlines() - читает строки и записывает в список
* f.write() – записывает
* f.writelines(lines) – записывет список строк
* f.truncate(size) – обрезает до указанного размера
* f.seek(offset) – передвигает курсор
* f.tell()  – возвращает место курсора

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

* Чтение:

f = open('…')

content = f.read()

lines\_list = f.readlines()

**for** i **in** range(5):

string = f.readline()

**for** s **in** f:

print(s)

f.close()

* Запись:

f = open('…')

f.write('…')

f.writelines(['…', '…'])

print('def', file=f)

f.close()

Файлы. Текстовые файлы. Итерационное чтение содержимого файла.

**with** open('...') **as** f:

s = f.readline()

**while** s.strip() != '':

s = f.readline()

**for** s **in** f:

print(s)

Файлы. Бинарные файлы. Основные методы. Сериализация данных.

* Бинарные файлы – файл, состоящий из последовательность произвольных байтов.
* Методы:
* close()
* read()
* write()
* tell()
* seek()
* truncate()
* fileno() – целочисленный дискриптор
* flush() – очищает внутренний буфер

**with** open('data.bin', 'rb') **as** f:

b = f.read(1):

**while** b != b'':

d = f.read()

* Сериализация данных – процесс перевода структуры данных в последовательность байтов:

s = struct.pack(format, objects)

Файлы. Оператор with. Исключения.

with open(…) as f1, open(…) as f2:

операторы

* Исключения:
* При открытии файла
* При записи
* При любых операциях

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

* Bytes и bytearray – классы для представления бинарных, байтовые строки.
* Создание:
* b'bytes'
* 'string'.encode('utf-8')
* bytes('string', encoding='utf-8')
* bytes([1, 2, 3, 4])
* bytearray(b’string') (массив)

Преобразование в обычную строку:

‘string'.decode('utf-8')

* bytes – неизменяемый, bytearray – изменяемый

Модуль struct.

* Формирует упакованные двоичные структуры данных из переменных базовых типов данных и распаковывает обратно.
* Функции:
* pack(format, v1, v2, …)
* pack\_into(format, buffer, offset, v1, v2, …) (записать в бувер)
* unpack(format, buffer)
* unfack\_from(format, /, buffer, offset=0) (начиная с позиции)
* iter\_unpack(format, buffer)
* calcsize(format) – размер формата
* Формат:
* @ – по умолчанию
* = порядок нативный, размер стандартный
* < порядок от младшего к старшему
* > от старшего к младшему
* ! сетевой
* s f, i

Модуль os. Основные функции.

* Модуль os предоставляет множество функций для работы с операционной системой, причём их поведение, как правило, не зависит от ОС, поэтому программы остаются переносимыми.
* os.path.exists(path) – существует путь
* os.path.getsize(path) – размер файла
* os.sep – разделитель пути
* os.getcwd()
* os.getpid() – текущий id процесса
* os.chdir() – смена текущей директории
* os.mkdir()
* os.remove() – удаляет путь к файлу
* os.rename()
* os.truncate()

Генераторы.

* Генератор возвращает итератор, по которому можно проходить пошагово, получая доступ к одному значению с каждой итерацией.
* Генераторы выражения: (i \*\* 2 for i in range(10))
* yield – ключевое слово, которое используется вместо return. yield временно приостанавливает исполнение, сохраняет состояние и затем может продолжить работу позже.

|  |
| --- |
| *# генерация нового списка, состоящего*  *# только из четных чисел*  **def** get\_even(list\_of\_nums) :  **for** i **in** list\_of\_nums:  **if** i % 2 == 0:  **yield** i    *# инициализация списка*  list\_of\_nums = [1, 2, 3, 8, 15, 42]    *# вывод начального списка*  print ("До фильтрации в генераторе: " + str(list\_of\_nums))    *# вывод только четных значений из списка*  print ("Только четные числа: ", end = " ")  **for** i **in** get\_even(list\_of\_nums):  print (i, end = " ") |

Модуль numpy. Обработка массивов с использованием данного модуля.

* numpy - библиотека языка Python ориентирована на работу с многомерными массивами и матрицами, с полиномами и с другими объектами.

**import** numpy **as** np

* Создание одномерного массива:

a = np.array((2, -3, 1, 9))

b = np.array([-7, 7, 8,4], dtype=float)  
c = np.array(list(map(float, input().split())))

* Создание двухмерного массива:

a = np.array([[1, 2, 3], [9, 8, 7]], 'int64')

* Создание матрицы из списка и обратная операция

lst = [[4, 5, 6], [7, 8, 9]]  
d = np.array(lst)

d1 = d.tolist()

* numpy.arange([start, ]stop, [step, ]dtype=**None**) – возвращает массив (numpy.ndarray) чисел, равномерно распределенных в заданном интервале.
* numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0) – возвращает массив (numpy.ndarray) N чисел, равномерно распределенных в заданном интервале.
* set\_printoptions(edgeitems=3, infstr='inf', linewidth=75, nanstr='nan', precision=8, suppress=**False**, threshold=1000, formatter=**None**) – позволяет настроить параметры вывода массивов на экран.
* edgeitems - количество элементов в начале и в конце каждой размерности,
* infstr - строковое представление inf,
* linewidth -число символов в строке, после которых производится перенос,
* nanstr - строковое представление NaN,
* precision - число цифр после запятой,
* suppress - если True, не печатает маленькие значения в научной нотации,
* threshold - число элементов в массиве, вызывающее обрезание,
* formatter - позволяет более тонко управлять печатью массива.
* Пустая матрица

a = np.empty((3, 2))

* Единичная матрица

b = np.identity(4)

* Нулевая матрица

c = np.zeros((4, 5), int)

* Матрица из единиц

d = np.ones((4,3), 'int64')

* Заполнена 7

f = np.full((3, 3), 7)

a = np.arange(12)

b = np.reshape(a,(2,6))

c = np.resize(a,(2,7)) # Изменение формы при несовпадении числа элементов

d = np.copy(a)

d.shape = (2,2,3) # Формирование 3х мерного массива

a = a.T

dsum = b.sum()

dmin = b.min()  
dmax = b.max()

mincol = b.min(axis=0)

maxrow = b.max(axis=1)

* Сортировка

sort(axis=-1, kind='quicksort', order=**None**)  
axis - ось, по которой идёт сортировка.По умолчанию по последней оси

kind - тип сортировки. Возможные значения 'quicksort', 'mergesort', 'heapsort'

c.sort(axis=0,kind='mergesort') *# Сортировка столбцов*  
c.sort(axis=1) # Сортировка строк

* Сортировка по определенным значениям:

**import** numpy **as** np

dtype = [('weight',int), ('height',float), ('age', int)]

v = [(70, 1.75, 15), (65, 1.8, 19), (45, 1.85, 16)]

c = np.array(v, dtype=dtype)

c.sort(axis=0, order = ['weight'])

print(c)

* Матричные операции:

c\_plus = a + b

c\_minus = a - b

c\_mult = a \* b

c\_power = a \*\* b

c\_num = a \* 5

c\_multmat = np.dot(a, b) # Матричное умножение

* Другой способ

d = np.add(a,b) # Сложение

e = np.subtract(a, b) # Вычитание

f = np.multiply(a, b) # Умножение

g = np.divide(a,b) # Деление

h = np.dot(a,b) # Матричное умножение

m = np.negative(a) # Замена знаков

n = np.transpose(a) # Тринспонирование

Модуль numpy. Работа с числами и вычислениями.

sum(axis = 1)

min(axis = 0)

max()

* Матричные операции:

c\_plus = a + b

c\_minus = a - b

c\_mult = a \* b

c\_power = a \*\* b

c\_num = a \* 5

c\_multmat = np.dot(a, b) # Матричное умножение

* Другой способ

d = np.add(a,b) # Сложение

e = np.subtract(a, b) # Вычитание

f = np.multiply(a, b) # Умножение

g = np.divide(a,b) # Деление

h = np.dot(a,b) # Матричное умножение

m = np.negative(a) # Замена знаков

n = np.transpose(a) # Тринспонирование

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

* Пакет Matplotlib является основным для визуализации расчетных данных.
* Для рисования графика используем модуль pyplot из пакета.

**import** matplotlib.pyplot **as** plt

x = [0, 6, 11, 17, 24, 31, 38, 40]

y = [0, 2, 13, 10, 16, 12, 20, 17]

plt.plot(x, y)

plt.show()

* Изменение цвета линии
* b, blue - синий
* k, black - черный
* w, white - белый
* c, cyan - голубой
* r, red - красный
* y, yellow - жёлтый
* g, green – зелёный
* m, magenta - малиновый

plt.plot( x, y, 'red')

plt.plot( x, y, ‘r')

* Изменение отрисовки линии:
* . точечный маркер
* , точки размером с пиксель
* o круги
* < треугольник вершиной влево
* > треугольник вершиной вправо s квадраты
* p пятиконечная звезда \*
* h шестиугольники
* + плюсы
* D ромбы
* d узкие ромбы
* | вертикальные линии

plt.plot( x, y, ‘p’)

* Типы линий:

- сплошная линия

-- штриховая линия

-. штрих – пунктирная

: пунктирная линия

Толщину линии нужно задавать аргументом linewidth(lw), а маркера markersize(ms). По умолчанию толщина равна 1.

* Заголовок и название осей:
* title(), можно указать параметр fontsize, horisontalaligment, verticaligment.
* xlabel(),
* ylabel().
* fontsize: 'large‘, ‘medium’, ‘small’.
* horisontalaligment(ha) :‘center’, ‘left’, ‘right’.
* verticaligment(va): ‘top’, ‘baseline’, ‘bottom’.
* Диапазон по осям

axis([0, 10, 0, 20])

Оси можно удалить командой axis(‘off’) и выравнить диапазоны по осям командой axis(‘equal’). При выравнивании диапазонов происходит не полное, а некоторое выравнивание. При этом axis(‘equal’) нужно задавать отдельной строкой. Предыдущие числа, заданные в axis([...]) не принимаются во внимание.

Для отображения координатной сетки grid(True).  
Функции xticks() и yticks() используются для дополнительного контроля шага координатной сетки.

Модуль matplotlib. Построение гистограмм и круговых диаграмм.

* Гистограмма используется для изображения зависимости частоты попадания элементов в соответствующий интервал группировки

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

x = np.random.normal(0, 3, 1000) # np.random.randn(1000)

plt.hist(x,25) # по умолчанию 10

plt.show()

* Круговые диаграмы – pie(), т. к. они похожи на разрезаемый пирог.
* Первым параметром является последовательность внесенных значений. После следуют необязательные аргументы:
* explode – часто кусок пирога выдвигают из центра. Эта последовательность имеет тот же размер, что и первый аргумент.
* colors - задает цвета. По умолчанию для matplotlib это blue, green, red, cyan, magenta, yellow.
* labels – это имена, у нас названия языков программирования.
* labeldistance – определяет радиальные расстояния, на котором эти имена выводятся
* auopct – задаёт, как форматируются численные значения
* pctdistance – каком расстоянии от центра располагаются числовые значения
* shadow – тень: boolean
* plt.axes(0.0, 0.0, 1.0, 1.0) - одинаковые размеры по осям.

|  |
| --- |
| **import** matplotlib.pyplot **as** plt  x = [6, 12, 20, 7, 5, 5]  languages = ['Matlab', 'Java', 'Python', 'C', 'C++', 'Other']  plt.figure(figsize=(10,10))  explode = [0, 0, 0, 0.1, 0, 0]  plt.pie(x, labels = languages, explode=explode, autopct='%1.1f%%', shadow=**False**)  plt.title('Circle diagram')  plt.show() |

Списки. Сортировка. Сортировка вставками. Сортировка выбором.

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

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

Списки. Сортировка. Метод быстрой сортировки.

|  |
| --- |
| # Метод простого выбора  # Перегоняем по одному минимальному элементу в начало списка  **def** selection\_sort(arr):  **for** i **in** range(len(arr)):  min\_ind = i  **for** j **in** range(i + 1,len(arr)):  **if** arr[j] < arr[min\_ind]:  min\_ind = j  arr[i],arr[min\_ind] = arr[min\_ind],arr[i]  **return** arr |

|  |
| --- |
| # Пузырьковая с флагом  # Сравниваем элементы попарно и максимальный перемещается в конец на каждом проходе  **def** bubble\_sort\_with\_flag(arr):  n = len(arr)  **for** i **in** range(n - 1):  flag = True  **for** j **in** range(n - 1 - i):  **if** arr[j] > arr[j + 1]:  arr[j],arr[j+1] = arr[j+1],arr[j]  flag = False  **if** flag:  **break**  **return** arr |

|  |
| --- |
| # Сортировка вставками  # В отсортированную часть вставляем элементы по очереди  **def** insertion\_sort(alist):  **for** i **in** range(1, len(alist)):  temp = alist[i]  j = i - 1  **while** (j >= 0 **and** temp < alist[j]):  alist[j + 1] = alist[j]  j = j - 1  alist[j + 1] = temp  **return** alist |

|  |
| --- |
| **def** shakersort(arr):  left = 0  right = len(arr) - 1  **while** left < right:  r\_new = left  **for** i **in** range(left,right):  **if** arr[i] > arr[i+1]:  arr[i], arr[i + 1] = arr[i + 1], arr[i]  r\_new = i  right = r\_new  l\_new = right  **for** i **in** range(right - 1, left - 1, -1):  **if** (arr[i] > arr[i+1]):  arr[i], arr[i + 1] = arr[i + 1], arr[i]  l\_new = i  left = l\_new  **return** arr |

|  |
| --- |
| # Вставки с бинарным поиском  **def** insertion\_sort\_with\_bin\_search(arr):  **for** i **in** range(1, len(arr)):  temp = arr[i]  left, right = 0, i  **while** left < right:  mid = (left + right)//2  **if** temp < arr[mid]:  right = mid  **else**:  left = mid + 1  j = i  **while** (j > left **and** j > 0):  arr[j] = arr[j-1]  j = j - 1  arr[left] = temp  **return** arr |

|  |
| --- |
| **def** shell(mas):  step = len(mas) // 2  **while** step:  **for** i, el **in** enumerate(mas):  **while** i >= step **and** el < mas[i - step]:  mas[i] = mas[i - step]  i -= step  mas[i] = el  step = 1 **if** step == 2 **else** int(step \* 5 // 11)  **return** mas |

|  |
| --- |
| # Шейкерная сортировка  **def** shakersort(arr):  left = 0  right = len(arr) - 1  **while** left < right:  r\_new = left  **for** i **in** range(left,right):  **if** arr[i] > arr[i+1]:  arr[i], arr[i + 1] = arr[i + 1], arr[i]  r\_new = i  right = r\_new  l\_new = right  **for** i **in** range(right - 1, left - 1, -1):  **if** (arr[i] > arr[i+1]):  arr[i], arr[i + 1] = arr[i + 1], arr[i]  l\_new = i  left = l\_new  **return** arr |

|  |
| --- |
| **import** random **as** r  # Быстрая сортировка  **def** quicksort(arr, start = 0, end = None):  **if** len(arr) == 0:  **return** arr  pind = r.randint(start,end-1)  pivot = arr[pind]  left = [x **for** x **in** arr **if** x < pivot]  midle = [x **for** x **in** arr **if** x == pivot]  right = [x **for** x **in** arr **if** x > pivot]  **return** quicksort(left, 0, len(left)) + [pivot] + quicksort(right, 0, len(right)) |