1. История развития вычислительных средств. Классификация ЭВМ

I поколение – основанное на электронных лампах, программирование на машинном языке.

II поколение – основанное на транизсторах, тогда началось использование ЯП и ОС

III поколнеие – основанное на интегральных схемах, введение многозадачности

IV поколение – основанное на больших и сверхбольших интегральных схемах, которые позволили развитие сетевых технологий, массовое распространение ПК.

V поколение – использование нейросетей, квантовых вычислений, облачные технологии и тп.

Классификация:

- по назначению: универсальные и специализированные
- по масштабу: суперЭВМ, мейнфреймы, мини-ЭВМ, микро-ЭВМ, микроконтроллеры
- по типу обработки данных: цифровые, аналоговые и гибридные
- по архитектуре: однопроцессорные, многопроцессорные, квантовые.

2. Представление чисел и форматы их хранения в ЭВМ.

ЭВМ оперирует только двоичными числами. Числа в памяти хранятся в разных форматах в зависимости от их типа данных.

Представление целых чисел состоит из бита знака и числового значения. Числовое значение представлено в прямом коде, двоичном виде числа, в случае положительного значения. В случае отрицательного значения, двоичное число инвертируется и преобразуется в дополнительный код.

Представление вещественных чисел делится на два формата, double и float, 62 битного и 32 битного представления.

В обоих случаях первый бит отводится под знак. Следующие 8 или 11 бит выделены под порядок (экспонента, по которому смещается точка плавающего числа). Оставшиеся биты (23 или 52) оставлены под мантиссу. Мантисса – это двоичное представление числа после плавающей точки.

3. Понятие архитектуры и структуры компьютера. Принципы фон Неймана.

Архитектура компьютера – это логическая организация компонентов компьютреа, то есть что делает каждая часть и как они взаимодействуют. Она описывает набор команд процессора, форматы данных и инструкций и обмен данными между компонентами.

Структура компьютера – это физическая реализация архитектуры, то есть как именно построены и связаны компоненты.

Принципы фор Неймана:

1. Программа и данные хранятся в одном типе памяти. Программа это те же даныые, которые можно менять
2. Каждая команда считывается из памяти и выполняется по очереди, одна за другой.
3. Центральный процессор управляет всем.
4. Оперативная память используется как для храннеия команд, так и для хранения данных.
5. Обмен с внешними устройствами осуществляется через команды, обрабатываемые процессором.

4. Основные компоненты ЭВМ. Основные типы архитектур ЭВМ.

Основные компоненты ЭВМ:

1. Центральный процессор
2. Оперативная память
3. Постоянная память
4. Устройства ввода
5. Устройства вывода

Основные типы архитектур ЭВМ

1. Архитектура фон Геймана: общая память программ и данных, последовательное выполнение команд.
2. Гарвардская архиектура – разделенная память для программ и данных.
3. Многопроцессорная архитектура: несколько параллельно работающих процессоров или ядер
4. Архитектуура с общей распределенной памятью
5. Архитектура RISC и CISC

5. Структура и классификация АСОИУ, виды обеспечения АСОИУ.

АСОИУ – автоматизированная система оперативного управления. Она предназначена для сбора, обработки и использования информации в реальном времени для управления производством, технологическими процессами или предприятием в целом.

Структура:

1. Уровень административного управления
2. Уровень координации
3. Уровень объекта

Классификация:

1. По сфере применения: промышленная, энергетическая, транспортная, логистическая.
2. По масштабу: локальная, региональная, отраслевая, межотраслевая
3. По степени автоматизации: информационные и информационно-управляющие.

Виды обеспечения АСОИУ

1. Техническое обеспечение
2. Программное обеспечение
3. Математическое обеспечение
4. Информационное обеспечение
5. Организационное обеспечение
6. Лингвистического обеспечение

6. Перечислите из каких этапов, с точки зрения программирования, состоит работа с любым файлом.

Этапы:

1. открытие файла
2. чтение и запись данных
3. обработка данных
4. закрытие файла

7. Объясните синтаксис функции open(), предназначенной для открытия файла. Поясните назначение параметров функции. Перечислите возможные значения режима Mode.

open() возвращает файловый дескриптор, соотвутствующий параметрам, использованным в функции.

Параметры open():

- file – путь к файлу
- mode – режим открытия
- encoding – кодировка файла
- buffering – уровень буферизации
- errors – способ обработки ошибок
- newline – управляет символами новой строки в текстовом режиме
- opener – функция, возвращающая открытый файловый дескриптор

Значения режима mode:

- r – read
- w – write
- a – append
- x – create

Дополнительные значения, которые добавляются к основным.

- b – binary
- t – text
- + - combining the operations

8. Объясните, каким образом происходит обработка ошибок, возникающих при работе с файлами. Приведите пример.

Ошибка при работе с файлами может произойти если: файл недоступен, поврежден или занят другим процессом. Обработка этих ошибок происходит с помощью try catch

```
try:
    with open("data.txt", "r", encoding="utf-8") as f:
        print(f.read())
except FileNotFoundError:
    print("Файл не существует.")
except PermissionError:
    print("Нет доступа к файлу.")
except Exception as e:
    print("Произошла ошибка при работе с файлом:", e)
```

9. Раскройте особенности методов read() и readline(). Приведите пример

- file.read(size) – считывает заданное количество байт всего файла для чтения в одну строку.
- file.readline(size) – считывает заданное количество байт одной строки файла для чтения

```
with open(“text.txt”, ‘r’, encoding=”utf-8) as f:
      text = f.read()
      line = f.readline();
```

10. Строки. Приведите примеры базовых алгоритмов строк.

Строки – последовательность символов, которая используется для представления текстовой информации.

Базовые алгоритмы: подсчет длины, поиск подстроки, замена подстроки, извлечение подстроки, разделение строки по символу и тп.

Примеры:

поиск

```
def contains(text, pattern):
    for i in range(len(text) - len(pattern) + 1):
        if text[i:i+len(pattern)] == pattern:
            return True
    return False

print(contains("hello world", "world"))
```

Разделение строки по символу

```
s = "apple,banana,cherry"
parts = s.split(',')
print(parts)  # ['apple', 'banana', 'cherry']
```

11. Строки. Основные функции для работы со строками.

Строки – последовательность символов, которая используется для представления текстовой информации.

Основные функции:

- len(string) – длина строки
- s.lower(), s.upper() - нижний верхний регистр
- s.replace(old, new) – заменя подстроки
- s.find(sub) поиск подстроки
- s.split() разделение строки по символу

12. Строки. Форматирование строк

Строки – последовательность символов, которая используется для представления текстовой информации.

Форматирования строк – это способ вставки переменных и значений внутрь строки с заданным форматом.

1. "Имя: %s, Возраст: %d" % (name, age) - %s – string, %d – digit, %f – float
2. "Имя: {}, Возраст: {}".format(name, age)
3. "Возраст: {1}, Имя: {0}".format(name, age)
4. "Имя: {n}, Возраст: {a}".format(n="Анна", a=20)
5. f"Имя: {name}, Возраст: {age}"

13. Форматный ввод/вывод. Спецификации формата: правила их записи и использования.

Форматный ввод/вывод — это способ точного управления отображением данных, например чисел, строк или дат, при выводе на экран или в файл.

1. "Имя: %s, Возраст: %d" % (name, age) - %s – string, %d – digit, %f – float
2. "Имя: {}, Возраст: {}".format(name, age)
3. "Возраст: {1}, Имя: {0}".format(name, age)
4. "Имя: {n}, Возраст: {a}".format(n="Анна", a=20)
5. f"Имя: {name}, Возраст: {age}"

14. Регулярные выражения. Основные функции Regex

Регулярные выражения – интсрумент поиска, сопоставления и обработки текста по шаблону.

Содержится в библиотеке **re**

1. re.search(pattern, string) – ищет первое совпадение с шаблоном в строке
2. re.match(pattern, string) – ищет совпадение только в начале строки
3. re.findall(pattern, string) – возвращает список всех совпадений
4. re.sub(pattern, repl, string) – заменяет все совпадения на заданную строку
5. re.compile(pattern) – компилирует шаблон для многократного использования (возвращает объект, к которому можно применять те же методы, что и к re, только с заранее заданным regex.

15. Регулярные выражения. Основные метасимволы в Regex

Регулярные выражения – интсрумент поиска, сопоставления и обработки текста по шаблону.

Они используют метасимволы – специальные символы, которые задают шаблон для поиска текста.

Метасимвол	Назначение
.	Любой **один** символ, кроме \n
^	Начало строки
$	Конец строки
*	0 или более повторений
+	1 или более повторений
?	0 или 1 повторение
{n}	Ровно n повторений
{n,}	n и более
{n,m}	От n до m повторений
[]	Один символ из набора
[^...]	Один символ **не** из набора
`	`
()	Группировка (захват группы)

16. Регулярные выражения. Флаги в Regex

Регулярные выражения – интсрумент поиска, сопоставления и обработки текста по шаблону.

Флаги – специальные параметры, которые заменяют поведение поиска

Флаг	Назначение
re.IGNORECASE или re.I	Игнорировать **регистр** (a = A)
re.MULTILINE или re.M	Многострочный режим: ^ и $ работают **в каждой строке**, а не только в начале и конце всей строки
re.DOTALL или re.S	Символ . начинает также **совпадать с символом новой строки** (\n)
re.VERBOSE или re.X	Позволяет писать шаблон с **комментариями и пробелами** для читаемости
re.ASCII или re.A	Интерпретирует \w, \d, \s как ASCII, а не Unicode

17. Регулярные выражения. Жадный и нежадный виды поиска.

Регулярные выражения – интсрумент поиска, сопоставления и обработки текста по шаблону.

Жадный поиск – выражение, захватывающее максимально взоможное количество символов, удовлетворяющих шаблону.

Нежадный поиск – захватывает минимально возможное количество символов, чтобы шаблон сработал.

Жадные квантификаторы:

Квантификатор	**Значение**
*	0 или более
+	1 или более
?	0 или 1
{m,n}	от m до n раз

Нежадные квантификаторы

Квантификатор	**Значение**
*?	0 или более, минимально
+?	1 или более, минимально
??	0 или 1, минимально
{m,n}?	от m до n, минимально

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

Файлы – именованные наборы данных, сохраненные во внешней памяти.

Программная обработка файлов – это открытие, чтение, запись, закрыти, обработка ошибок.

Дескриптор файла – это числовой или объектный идентификатор, который операционная система или среда программирования использует для управления открытым файлом.

Виды файлов: текстовые, бинарные(exe, jpg, mp3) , специализированные (архивы, базы данных и тп)

19. Файлы. Режимы доступа к файлам.

Файлы – именованные наборы данных, сохраненные во внешней памяти.

Режимы доступа:

Режим	Описание
'r'	Открыть для чтения (файл должен существовать)
'w'	Открыть для записи (если файл есть — очистить, если нет — создать новый)
'a'	Открыть для добавления (запись в конец файла, если нет — создать)
'x'	Создать новый файл для записи (если файл существует — ошибка)
'b'	Открыть в **бинарном** режиме (добавляется к другим режимам)
't'	Открыть в **текстовом** режиме (по умолчанию)
'+'	Открыть для чтения и записи одновременно

20. Файлы. Текстовые файлы. Основные методы для работы.

Файлы – именованные наборы данных, сохраненные во внешней памяти.

Текстовые файлы – файлы, данные в которых хранятся в символьном виде.

Методы:

- open(filepath, mode) – получения дескриптора
- read(size) - чтение всего файла или заданного количества
- readline(size) – чтение заданного количества символов строки
- write(string) – запись строки
- writelines(list_of_strings) – запись списка строк
- close() закрытие файлового дескриптора

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

Файлы – именованные наборы данных, сохраненные во внешней памяти.

Текстовые файлы – файлы, данные в которых хранятся в символьном виде.

Чтение и запись происходят посредством следующих методов:

- open(filepath, mode) – получения дескриптора
- read(size) - чтение всего файла или заданного количества
- readline(size) – чтение заданного количества символов строки
- write(string) – запись строки
- writelines(list_of_strings) – запись списка строк
- close() закрытие файлового дескриптора

Поиск происхоит последством параллельного чтения строки и поиска с использованием метдов поиска подстрок.

22. Файлы. Текстовые файлы. Итерационное чтение содержимого файла

Файлы – именованные наборы данных, сохраненные во внешней памяти.

Текстовые файлы – файлы, данные в которых хранятся в символьном виде.

Для итерационного чтения запускается построчное чтение с помощью цикла for

```
with open('file.txt', 'r', encoding='utf-8') as f:
    for line in f:
        print(line.strip())
```

23. Модуль os. Основные функции

Модуль os предоставляет функции для взаимодействия с операционной системой: работа с файлами, каталогами, процессами и др.

Основные функции модуля

Функция	**Описание**
os.getcwd()	Возвращает текущий рабочий каталог
os.chdir(path)	Изменяет текущий рабочий каталог
os.listdir(path='.')	Возвращает список файлов и папок в каталоге
os.mkdir(path)	Создаёт новый каталог
os.makedirs(path)	Создаёт вложенные каталоги (включая промежуточные)
os.remove(path)	Удаляет файл
os.rmdir(path)	Удаляет пустой каталог
os.rename(src, dst)	Переименовывает файл или каталог
os.stat(path)	Получает информацию о файле/каталоге

24. Модуль os. Обход дерева каталогов.

Модуль os предоставляет функции для взаимодействия с операционной системой: работа с файлами, каталогами, процессами и др.

Обход дерева – os.walk() - возвращает кортежи вида (dirpath, dirnames, filenames) после рекурсивного обходя дерева каталогов.

25. Модуль os.path. Обработка абсолютных и относительных путей доступа

Модуль os предоставляет функции для взаимодействия с операционной системой: работа с файлами, каталогами, процессами и др.

Абсолютный путь – польный путь от корня файловой системы

Относительный путь – путь относительно текущего каталога

Методы обработки путей:

Функция	Описание
os.path.abspath(path)	Возвращает абсолютный путь
os.path.relpath(path, start)	Возвращает относительный путь от start до path
os.path.isabs(path)	Проверяет, является ли путь абсолютным
os.path.join(path, *paths)	Склеивает путь корректно для ОС
os.path.normpath(path)	Нормализует путь (убирает лишние .., . и /)
os.getcwd()	Возвращает текущий рабочий каталог

26. Модуль zipfile. Создание и добавление ZIP-файлов

Модуль zipfile – позволяет работать с архивами ZIP: создание, извлечение и просмотр содержимого.

Создание

```
with zipfile.ZipFile('archive.zip', mode='w') as zipf:
    zipf.write('file1.txt')  # добавляем файл
    zipf.write('file2.txt')  # добавляем ещё один
```

Добавление

```
with zipfile.ZipFile('archive.zip', mode='a') as zipf:
    zipf.write('file3.txt')
```

27. Модуль zipfile. Сжатие файлов

Модуль zipfile – позволяет работать с архивами ZIP: создание, извлечение и просмотр содержимого.

Сжатие файлов

```
import zipfile

with zipfile.ZipFile('compressed.zip', 'w', compression=zipfile.ZIP_DEFLATED) as
zipf:
    zipf.write('file1.txt')
    zipf.write('file2.txt')
```

Алгоритмы сжатия:

Алгоритм	Константа
Без сжатия	zipfile.ZIP_STORED
DEFLATE	zipfile.ZIP_DEFLATED
BZIP2	zipfile.ZIP_BZIP2
LZMA	zipfile.ZIP_LZMA

28. Модуль zipfile. Чтение ZIP-файлов.

Модуль zipfile – позволяет работать с архивами ZIP: создание, извлечение и просмотр содержимого.

Открытие и просмотр содежимого

```
import zipfile

with zipfile.ZipFile('example.zip', 'r') as zipf:
    print("Содержимое архива:")
    print(zipf.namelist())
```

29. Модуль zipfile. Извлечение файла из ZIP-архива

Модуль zipfile – позволяет работать с архивами ZIP: создание, извлечение и просмотр содержимого.

Основные методы извлечения:

один файл:

```
with zipfile.ZipFile('archive.zip', 'r') as zipf:
    zipf.extract('file1.txt', path='extracted/')
все:
with zipfile.ZipFile('archive.zip', 'r') as zipf:
    zipf.extractall('unzipped/')
указанные файлы:
with zipfile.ZipFile('archive.zip', 'r') as zipf:
    zipf.extractall(path='target/', members=['file1.txt', 'folder/file2.txt'])
```

30. Модуль shutil. Архивация

Модуль shutil – предоставляет высокоуровневые функции для работы с файлами, дикерторями и архивами – копирование, удаление, перемещение и др.

shutil.make_archive(base_name, format, root_dir)

Поддерживаемые форматы:

Формат	Описание
'zip'	ZIP-архив
'tar'	tar без сжатия
'gztar'	tar + gzip (.tar.gz)
'bztar'	tar + bzip2 (.tar.bz2)
'xztar'	tar + xz (.tar.xz)

31. Модуль shutil. Операции над файлами и директориями

Модуль shutil – предоставляет высокоуровневые функции для работы с файлами, дикерторями и архивами – копирование, удаление, перемещение и др.

- shutil.copy(src, dst) копирование
- shutil.copy2(src, dst) копирование с метаданными (не только содержимое, но и описание файлов
- shutil.copytree(src, dst) копирование всей дикертории
- shutil.rmtree(path) удаление директории
- shutil.move(src, dst) перемещение
- shutil.getsize(path) возвращает размер файла в байтах

32. Модуль shutil. Копирование файлов и папок

Модуль shutil – предоставляет высокоуровневые функции для работы с файлами, дикерторями и архивами – копирование, удаление, перемещение и др.

- shutil.copy(src, dst) копирование
- shutil.copy2(src, dst) копирование с метаданными (не только содержимое, но и описание файлов
- shutil.copytree(src, dst) копирование всей дикертории

33. Модуль shutil. Перемещение и переименование файлов и папок.

Модуль shutil – предоставляет высокоуровневые функции для работы с файлами, дикерторями и архивами – копирование, удаление, перемещение и др.

shutil.move(src, dst) перемещает папку или файл в новое место по dst

34. Модуль pathlib. Основные классы модуля

Модуль pathlib – позволяет обращаться с путями как с объектами, а не строками.

Основные классы

pathlib.Path – универсальный путь, автоматически выбирает подходящий подкласс для ОС.

```
from pathlib import Path

p = Path('example.txt')
print(p.exists())
```

Path.PurePath – базовый класс, не зависит от операционной системы. Предназначен для манипуляции строками путей, но не обращается к файловой системе

```
from pathlib import PurePath

p = PurePath('folder', 'file.txt')
print(p)
```

35. Модуль logging. Характеристики уровней логирования

Модуль logging позволяет выводить **диагностические сообщения** (логи) во время работы программы.

Уровни логирования

Уровень	Значение	Назначение
DEBUG	10	Подробная информация для отладки. Используется разработчиками.
INFO	20	Общая информация о ходе выполнения программы.
WARNING	30	Внимание: потенциальная проблема или нештатная ситуация, но программа работает.
ERROR	40	Ошибка: программа не смогла выполнить какую-то операцию.
CRITICAL	50	Критическая ошибка: программа может быть остановлена.

При настройке логгера указывается минимальный уровень, при котором уровни нижу будут игнорироваться