Понятие программы и программирования. Виды программного обеспечения.

Программа (program, routine) – упорядоченная последовательность команд (инструкций) компьютеру для решения задачи.

Программи́рование — **процесс создания компьютерных программ**.

Все программы можно разделить на три категории: **системное программное обеспечение (ОС, драйверы, дополнения); прикладное программное обеспечение (графич и текстовые редакторы, браузеры, базы данных и т.д.); инструментальное программное обеспечение (компиляторы, отладчики, переводчики высокого уровня, редакторы, интерпретаторы)**.

Жизненный цикл программного обеспечения. Стадии разработки программ.

Жизненный цикл программного обеспечения (ПО) — **период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации**

Стадии разработки ПО:

1. **Анализ предметной области (постановка задачи);**
2. **Разработка проекта системы: Создание модели, отражающей основные функциональные требования, предъявляемые к программе; Выбор метода решения (построение математической модели); Разработка алгоритма – последовательности действий по решению задачи;**
3. **Реализация программы на языке программирования (кодирование);**
4. **Тестирование ПО;**
5. **Внедрение и сопровождение.**

Программная документация и стандарты на разработку прикладных программных средств.

Единая система программной документации - комплекс государственных стандартов, устанавливающих взаимосвязанные правила разработки, оформление и обращения программ и программной документации.
В стандартах ЕСПД устанавливают требования, регламентирующие разработку, сопровождение, изготовление и эксплуатацию программ.

В состав ЕСПД входят:

- основополагающие стандарты;
- стандарты, определяющие формы и содержание программных документов;
- стандарты, обеспечивающие автоматизацию разработки программных документов.

Понятие алгоритма. Основные требования, предъявляемые к алгоритмам.

Алгоритм — точное описание порядка действий, которые должен выполнить исполнитель для решения задачи за конечное время.

Требования к алгоритмам – понятность (в алгоритм должны входить только команды из системы команд исполнителя), корректность (при допустимых исходных данных должен получаться правильный результат), наличие механизма реализации, массовость (возможность применения алгоритма к кругу однотипных задач с разными исходными данными), эффективность (затрата ограниченного числа ресурсов на выполнение).

Способы описания алгоритмов.

(словесный) естественный язык, псевдокод, блок-схема (граф-схема)
Псевдокод – описание структуры алгоритма на естественном, но частично формализованном языке. В псевдокоде используются некоторые формальные конструкции и общепринятая математическая символика.

Основные типы управляющих конструкций. Операторы выбора.

Управляющие конструкции языка - это наборы служебных слов, позволяющие изменять ход выполнения скрипта.
if… else…, неполная форма - if…,
операторы сравнения, логические операторы,
множественный выбор (match: case…)
Для решения задач выбора применяются:

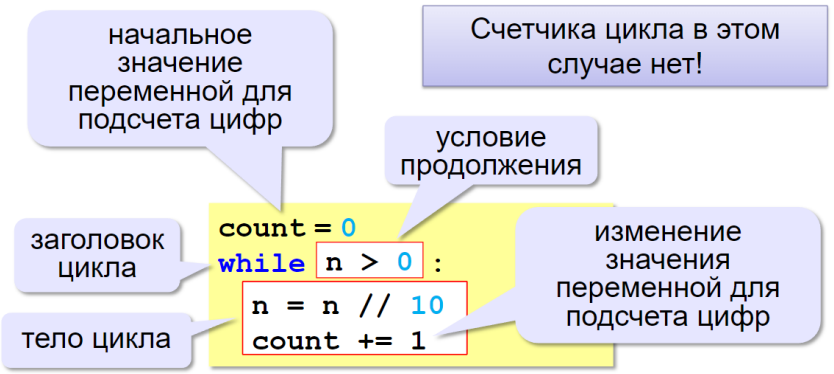
1. **Операции сравнения**, которые возвращают True (истина) или False (ложь) (<,>,<=,=>...)
2. **Логические операторы**, применяемые для проверки одновременно несколько условий(and, or, not…)

Основные типы управляющих конструкций. Циклы.

Цикл – это многократное выполнение одинаковых действий. Тело цикла может не выполняться ни разу.

While Предписывает выполнять тело цикла пока выполняется условие р.

For Выполняет инструкции для каждого элемента коллекции



Рекурсивные и итерационные алгоритмы.

Под итерацией в общем случае понимают такой **способ организации обработки данных, при котором многократно повторяется некоторая последовательность действий**.

Алгоритм, основу которого составляет итерация, называется *итерационным* (*итеративным*). **Примером итерационных алгоритмов являются циклические конструкции.**

Обобщением понятия итерации является **рекурсия**, под которой понимают т**акой способ организации обработки данных, при котором алгоритм использует сам себя в качестве вспомогательного алгоритма**. Классическими примерами рекурсивных алгоритмов являются вычисление факториала и чисел Фибоначчи.

Методы сортировки данных.

Сортировка пузырьком считается самым простым алгоритмом сортировки. В данном случае алгоритм проходит по массиву несколько раз, причём на каждом этапе происходит перемещение в конец массива самого большого из неотсортированных значений.
Сортировка данных вставками работает путём прохождения по массиву и перемещения нужного значения в его начало. Важно помнить, что сортировка обрабатывает элементы массива по порядку. Всё, что находится слева от текущего индекса, — отсортировано.
Сортировка выбором —некий гибрид между сортировкой вставками и пузырьковой сортировкой. Это алгоритм сортировки массивов, в котором на каждой итерации во всей последовательности не отсортированных данных выбирается минимальный элемент (при сортировке по возрастанию) и помещается в первую позицию не отсортированной последовательности.
Быстрая сортировка тоже представляет собой алгоритм, отвечающий типу «разделяй и властвуй». Сортировка данных происходит рекурсивно и поэтапно:
1) Выбирается ключевой индекс, массив делится по нему на 2 части. **2)** Все элементы, которые больше ключевого, перемещаются в правую часть массива, которые меньше — в

левую. Теперь ключевой элемент больше любого значения слева и меньше любого значения справа. **3)** Первые два шага повторяются до полной сортировки массива.

Простые типы данных

К простым типа данных относят числа и строки.

Числа бывают 3 видов: целые числа(int), числами с плавающей точкой - вещественные (float) и комплексные числа (complex).

Строки (string) — это последовательности символов, поэтому к ним применимы многие методы других последовательностей. Например, обращение к элементу по индексу, вычисление количества символов, конкатенация и получение среза.

Структурные типы данных. Структура данных «очередь».

К структурным типам данных относятся: списки, кортежи, словари, множества, очереди и др.

Список – это структура данных, которая содержит упорядоченный набор элементов, т.е. хранит *последовательность* элементов.

Кортежи служат для хранения нескольких объектов вместе. Их можно рассматривать как аналог списков, но без такой обширной функциональности, которую предоставляет класс списка.

Словарь – это некий аналог адресной книги, в которой можно найти адрес или контактную информацию о человеке, зная лишь его имя; т.е. некоторые **ключи** (имена) связаны со **значениями** (информацией)

Множества – это *неупорядоченные* наборы простых объектов. Они необходимы тогда, когда присутствие объекта в наборе важнее порядка или того, сколько раз данный объект там встречается.

Структура данных “очередь” - это линейная структура данных, в которой данные хранятся в порядке «первым пришёл — первым ушёл» (FIFO). В отличие от массивов, вы не можете получить доступ к элементам по индексу, а вместо этого можете извлечь только следующий самый старый элемент.

Структура данных «стек».

Стек - такой последовательный список с переменной длиной, включение и исключение элементов из которого выполняются только с одной стороны списка, называемого вершиной стека.

Основные операции над стеком: включение нового элемента(**push**), исключение элемента из стека(**pop**)

Алгоритмическая декомпозиция.

Алгоритмическая декомпозиция - это принцип функционального программирования, согласно которому задачу (сложные или повторяющиеся действия) разбивают на отдельные функции (подпрограммы), которые выполняют отдельные конкретные шаги, действия поставленной программной задачи.

Императивный (процедурный) подход к проектированию программ. Понятия функции.

Императивный подход к проектированию программ построен на принципе **алгоритмической декомпозиции**, т е заключается в том, что любая задача, которая решается программой, должна быть разбита на как можно более элементарные последовательные и, часто независимые, однозначные шаги (алгоритмы), которые должна выполнить программа.

Именно этот подход вводит понятие **функции** в программу - подпрограммы / части программы, которые выполняют небольшой алгоритм при их вызове и снабжении информацией из другой (часто главной) части программы.

Принципы структурного проектирования.

Сущность структурного подхода к разработке ПО ИС заключается в ее **декомпозиции на автоматизированные функции**: система разбивается на функциональные подсистемы, которые в свою очередь делятся на подфункции, они - на задачи и так до конкретных процедур. При этом ИС сохраняет целостность представления, где все составляющие взаимосвязаны.

- **принцип** абстрагирования - выделение существенных аспектов системы и отвлечение от несущественных;
- **принцип** непротиворечивости — обоснованность и согласованность элементов системы;
- **принцип** структурирования данных — данные должны быть структурированы и иерархически организованы.

Межмодульное взаимодействие.

Модуль в программировании — это фрагмент кода, имеющий определенное функциональное значение и характеризующийся логической завершенностью.

Модульное программирование — это способ создания программы посредством объединения модулей в единую структуру. Это - один из архитектурных подходов создания программы.

Модульное программирование крайне эффективно при групповых разработках, где каждый сотрудник может сконцентрироваться только на своём фронте работ.

Сущность функционального программирования.

Функциональное программирование определяют как программирование, в котором нет побочных эффектов.

Отсутствие побочного эффекта означает, что функция:

- полагается только на данные внутри себя,
- не меняет данные, находящиеся вне функции.

Вычисленный результат – есть единственный эффект выполнения любой функции. Функциональный код не полагается на данные вне текущей функции, и не меняет данные, находящиеся вне функции.
В Python для создания функциональных функций можно использовать анонимные функции lambda

Структура программы на языке высокого уровня.

Конкретная структура программы определяется языком программирования.

В общем виде программа включает следующие компоненты:

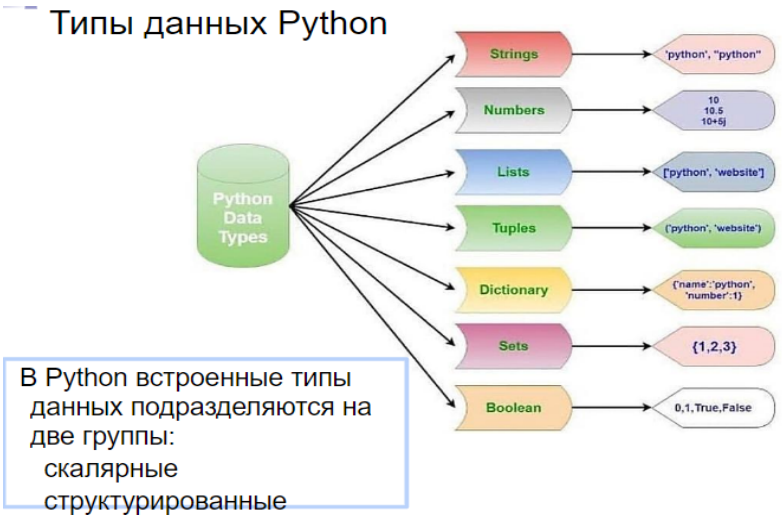
- список модулей, используемых в программе
- раздел описания констант
- раздел описания типов
- раздел описания переменных
- описание функций
- основной блок программы, включающий операторы

Типы данных языка высокого уровня.

Тип данных определяет формат представления в памяти компьютера, множество допустимых значений для переменной/константы, множество допустимых операций.

Типы данных можно разделить на:

- Скалярные (примитивные): Целочисленные, вещественные, логические и символьные
- Структурированные: Строка, массив, структура, перечисление, класс
  В python:



Реализация линейных вычислений. Операторы языка высокого уровня.

Операторы:

- ‘+’ - сложение
- ‘*’ - умножение
- ‘/’ - деление
- ‘//’ - целочисленное деление (округление вниз)
- ‘**’ - возведение в степень
- ‘^’ - побитовое сложение (xor)
- ‘%’ - остаток от деления (в python остаток >= 0 даже для отриц. чисел)

Сокращенные записи операций:

```
a += b  # a = a + b
a -= b  # a = a - b
a *= b  # a = a * b
a /= b  # a = a / b
a //= b # a = a // b
a %= b  # a = a % b
```

Операторы управления в языке высокого уровня.

К операторам управления относят:

- следование (vvod, изменение данных, vivod);
- ветвление (if/else, case);
- цикл (for, while, do while);
- передача управления (break, continue).

Операторы цикла в языке высокого уровня.

Организация цикла

- Цикл с известным числом повторений
- Цикл с неизвестным числом повторений

Управляющие структуры

- «цикл while» (пока логическое утверждение истинно)
- «цикл for» (для диапазона)
- «цикл foreach» (Выполняет инструкции для каждого элемента коллекции)

В языке Python циклы предлагают дополнительную возможность: сразу после повторно исполняемого внутреннего блока цикла можно поместить блок else

Организация программы с помощью функций в языке высокого уровня.

Разделение сложных и больших программ на небольшие легко управляемые части, называемые функциями называется организацией программы с помощью функций.

Если программе необходимо выполнить какую-либо задачу, то она вызывает соответствующую функцию, обеспечивая эту функцию информацией, которая ей понадобится в процессе обработки.

В Python можно создать четыре типа функций: 1) глобальные функции 2) локальные функции 3) лямбда-функции 4) методы

Описание и вызов функции в языке высокого уровня.

Описание функции в Python

def <имя функции> (<список параметров>):
<тело функции>
return

def действует как оператор присваивания: создаются новый объект-функция и ссылка на объект с указанным именем, которая указывает на объект-функцию

- тип - определяет тип значения, которое возвращает функция с помощью оператора return, по умолчанию функция возвращает значение типа int.
- список параметров - состоит из перечня типов и имен параметров, разделенных запятыми, круглые скобки обязательны.
- тело функции – набор выражений.

Области действия переменных в языке высокого уровня.

По зоне видимости различают **локальные** и **глобальные** переменные. Первые доступны только конкретной подпрограмме, вторые — всей программе.

Существует также и цепь областей видимости, когда программа пытается при вложенности функций найти переменную как можно раньше в функциях, постепенно переходя к поиску в глобальной области видимости.

Списки и их применение.

Список - структура данных, предназначенная для хранения значений (могут быть различного типа)

Списки в Python представляют собой непрерывные массивы ссылок на другие объекты. Индексация в списках начинается с нуля. Списки можно создавать с помощью генератора, например: b = [i+10 for i in a] . Используют списки для хранения различной информации, их можно соотносить поэлементно через индексы.

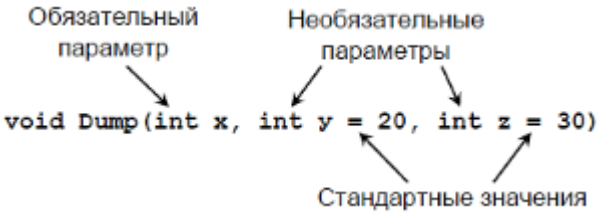
Кортежи и их применение.

Кортеж, по сути, является неизменяемым списком. Его удобно использовать в тех случаях, в которых данные намеренно организованы таким образом, чтобы их нельзя было

изменить, например, в качестве защиты от пользователя.При возврате из функции нескольких значений они также пакуются в кортеж. Создать кортеж: a = tuple(), a = ().

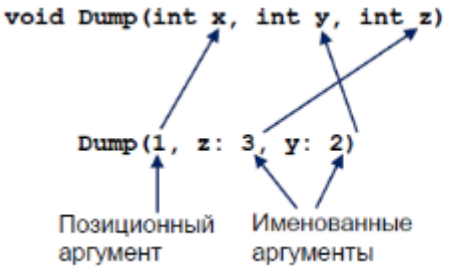
Реализация передачи параметров в функцию в языке высокого уровня.

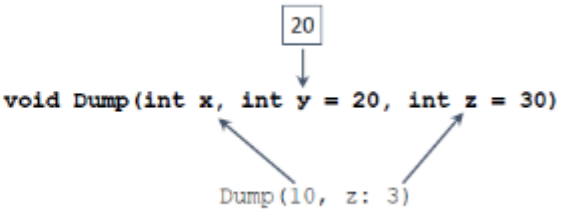
Объявление необязательных параметров разрешает пропуск их при указании аргументов. Все необязательные параметры должны располагаться после обязательных параметров.




```
>>> def func(a, b, c=2): # c - необязательный аргумент
...     return a + b + c
...
>>> func(1, 2)  # a = 1, b = 2, c = 2 (по умолчанию)
5
>>> func(1, 2, 3)  # a = 1, b = 2, c = 3
6
```

При передаче значения аргумента можно также указать имя параметра, для которого предназначено это значение. Компилятор проверяет, есть ли параметр с таким именем, и применяет для него заданное значение. Все именованные аргументы должны располагаться после позиционных аргументов.





Применение документирования функций в языке высокого уровня.

"""
Комментарий для документирования модуля и функции
"""

Словари и их применение.

Словари - неупорядоченная коллекция пар элементов “ключ-значение”. Относятся к структурированному типу данных. Изменяемы: можно изменять, добавлять и удалять элементы (пары "ключ:значение"). Словари могут быть заданы несколькими способами:

```
d1 = dict({"id": 1948, "name": "Washer", "size": 3})
d2 = dict(id=1948, name="Washer", size=3)
d3 = dict([("id", 1948), ("name", "Washer"), ("size", 3)])
d4 = dict(zip(("id", "name", "size"), (1948, "Washer", 3)))
d5 = {"id": 1948, "name": "Washer", "size": 3}
```

Также можно создать пустой словарь d = {}. Используются, когда необходимо создать *гибкую структуру данных, обеспечивающую возможность быстрого поиска.*

Реализация функционального программирования в языке высокого уровня

Тут говорим о функциях высшего порядка - это когда мы в функцию можем передать в качестве параметра другую функцию.
Обязательно упоминаем о лямбда-функциях - анонимных функциям. пример:
list(filter(lambda x: x > 0, [-11, 11, -21, 21, 0])) вернёт [11, 21]

Основы ООП. Реализация класса в языке высокого уровня.

Объектно-ориентированное программирование – это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Особенности:

- ООП использует в качестве базовых элементов объекты, а не алгоритмы
- каждый объект является экземпляром какого-либо определенного класса (играющего роль типа данных)
- классы организованы иерархически

Класс — это структура данных, объединяющая состояние (поля) и действия (методы и другие функции-члены).
Класс определяет:

- данные (переменные)
- поведение (методы)

Класс рассматривается как определяемый пользователем тип данных.
Объектом называется экземпляр некоторого класса. Объект создается как переменная типа класса, которая используется для доступа к данным - членам класса и для вызова методов - членов класса.

Назначение наследования и пример реализации этого механизма в языке высокого уровня.

Наследование — механизм, который позволяет описать новый класс на основе существующего (родительского). При этом свойства и функциональность родительского класса заимствуются новым классом. Для чего нужно наследование?

Прежде всего — чтобы не допустить повторное использование кода. Поля и методы, описанные в родительских классах, можно использовать в классах-потомках.

Реализация инкапсуляции в языке высокого уровня.

Инкапсуляция - это упаковка данных и функций в один компонент (например, класс) и последующий контроль доступа к этому компоненту, создавая "чёрный ящик" из объекта. По этой причине, пользователю необходимо знать только интерфейс этого класса (то есть данные и функции, предоставляемые для взаимодействия с классом извне), а не то, как он реализован внутри.

Python 3 предоставляет 3 уровня доступа к данным:

- публичный (`public`, нет особого синтаксиса, `publicBanana`);
- защищенный (`protected`, одно нижнее подчеркивание в начале названия, `_protectedBanana`);
- приватный (`private`, два нижних подчеркивания в начала названия, `__privateBanana`).