***Лабораторная работа №15***

**Выбор и обоснование выбора среды разработки программы. Изучение различных стилей программирования,  правил формирования листинга программы.**

**Цель работы:**

1. Изучить критерии выбора языка программирования. Научиться обосновывать выбор среды разработки в  соответствии с критериями выбора языка программирования.

2. Изучить основные парадигмы программирования.

3. Изучить правила формирования листинга программы.

**Порядок выполнения работы:**

1. Выполните задания 1-3.

**Задание 1. Критерии выбора языка программирования.**

Для разработки моего проекта я выбрал язык программирования python и стороннюю библиотеку pygame по следующим критериям:

-Объем потребляемых ресурсов.

-Скорость работы конечного продукта.

-Скорость разработки программы.

-Ориентированность на компьютер или человека.

-Скорость внесения изменений, скорость тестирования.

**Задание 2. Изучение парадигм программирования**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Название парадигмы** | **Сущность  парадигмы,**  **основные**  **идеи,**  **принципы,**  **объекты.** | **Языки**  **поддерживаю щие данную  парадигму.**  **(2-3 языка)** | **Достоинства** | **Недостатки** | **Примеры**  **программ** |
| **1.** | **Императивное**  **программирование** | В исходном коде записываются «приказы» команды, а не классы  Все инструкции должны выполняться последовательно, один за другим.  После выполнения инструкций данные могут записываться в память и считываться памяти. | Ассамблер, fortran, algo | эффективная реализация | Нет возможности перескакивать с одного участка кода на другой | Python  L = [1, 2, 3, 4, 5]  sum = 0  for x in L:  sum += x |
| **2.** | **Декларативное**  **программирование** | Декларативная программа состоит из ограничений и правил, из которых компьютер генерирует способ получения результата | SQL, MySQL | позволяет писать более безопасный и поддерживаемый код, который легко параллелится. А компиляторы декларативных языков имеют больше возможностей при оптимизации программ. | Сложно в понимании |  |
| **3.** | **Структурное**  **программирование** | представление программы в виде иерархической структуры блоков. | C, Pascal, Basic | позволяет сократить число вариантов построения программы по одной и той же спецификации  логически связанные операторы находятся визуально ближе, а слабо связанные — дальше  Сильно упрощается процесс тестирования и отладки структурированных программ. | работа со структурными данными практически не возможна без использования скриптов и знания как эти скрипты функционируют | **Pascal**  **Program** arr\_2;  **var** m, n, sum : integer;  arr : **array** [1..10, 1..10] **of** integer;  **begin**  **repeat**  writeln('Введите размер двумерного массива');  read(m);  read(n);  **until** (m > 0) **and** (m <= 10) **and** (n > 0) **and** (n <= 10);  sum := 0;  **for** m := 1 **to** m **do begin**  **for** n := 1 **to** n **do begin**  // randomize;  arr[m, n] := random(50);  write(arr[m, n] : 4);  **if** (m **mod** 2 = 0) **then**  sum := sum + arr[m, n];  **end**;  writeln();  **end**;  writeln('Сумма элементов равна = ', sum);  **end**. |
| **4.** | **Процедурное**  **программирование** | процедура может иметь несколько точек выхода (return в C-подобных языках), несколько точек входа, иметь аргументы, возвращать значение как результат своего выполнения, быть перегруженной по количеству или типу параметров и много чего еще. | C, C++, Pascal | отлично подходит для программирования общего назначения  Закодированная простота наряду с простотой реализации компиляторов и интерпретаторов  Исходный код переносим  Код может быть повторно использован в разных частях программы, без необходимости копировать его  Благодаря методике процедурного программирования требования к памяти также сокращаются  Ход программы можно легко отследить | Код программы труднее писать  Процедурный код часто не может быть использован повторно, что может привести к необходимости воссоздания кода, если это необходимо для использования в другом приложении.  Сложно общаться с реальными объектами  Данные открыты для всей программы, что делает их не очень безопасными | **Pascal**  **Program** zadproc\_1;  **var** x1, y1, x2, y2, dl1, dl2, dl3, m1, a, b, m : real;  **procedure** dlina(x1, y1, x2, y2 : real; **var** dl : real);  **begin**  dl := sqrt(sqr(x2 - x1) + sqr(y2 - y1));  **end**;    **procedure** mx(a, b : real; **var** m : real);    **begin**  **if**(a > b) **then**  m := a **else**  m := b;  **end**;  **begin**  writeln('Введите 1-ю коордитату');  readln(x1);  writeln('Введите 2-ю коордитату');  readln(x2);  writeln('Введите 3-ю коордитату');  readln(y1);  writeln('Введите 4-ю коордитату');  readln(y2);  writeln('Введите a');  readln(a);  writeln('Введите b');  readln(b);  dlina(x1, y1, 0, 0, dl1);  writeln(dl1);  dlina(x2, y2, 0, 0, dl2);  writeln(dl2);  dlina(a, b, 0, 0, dl3);  writeln(dl3);  mx(dl1, dl2, m1);  writeln('Сравнение первых 2 длин ', m1);  mx(dl3, m1, m);  writeln('Минимальное расстояние ', m);  **end**. |
| **5.** | **Модульное**  **программирование** | Программа описанная в стиле модульного программирования — это набор модулей. Что внутри, классы, императивный код или чистые функции — не важно. | Haskell, Pascal | Повышение удобства сопровождения кода.  Совместная работа нескольких человек не мешает друг другу. | невозможно загрузить модули по запросу. | **Pascal**  **Program** lab\_12;  **uses** lab\_12;  **var** x1, y1, x2, y2, dl1, dl2, dl3, m1, a, b, m, z : real;  **begin**  writeln('Введите 1-ю коордитату');  readln(x1);  writeln('Введите 2-ю коордитату');  readln(x2);  writeln('Введите 3-ю коордитату');  readln(y1);  writeln('Введите 4-ю коордитату');  readln(y2);  writeln('Введите a');  readln(a);  writeln('Введите b');  readln(b);  dlina(x1, y1, 0, 0, dl1);  writeln(dl1);  dlina(x2, y2, 0, 0, dl2);  writeln(dl2);  dlina(a, b, 0, 0, dl3);  writeln(dl3);  mx(dl1, dl2, m1);  writeln('Сравнение первых 2 длин ', m1);  mx(dl3, m1, m);  writeln('Минимальное расстояние ', m);  a:= 2.5;  b:= -7.3;  z := f(a, b) + f(sqr(a), sqr(b)) + f(sqr(a) - 1, b) + f(a - b, b) + f(sqr(a) + sqr(b), sqr(b) - 1);  write('Z = ', z);  **end**. |
| **6.** | **Объектно**  **ориентированное**  **программирование** | основанно на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования | Java, Python, C# | **Параллельная разработка, Модульность, Безопасность. Возможность повторного использования** | Низкая **производительность**  **Требуется больше планирования** | Python  class MyClass:  x = int(input('Введите значение '))    def print\_x(self):  print(self.x)    x1 = MyClass()  x1.print\_x() |
| **7.** | **Функциональное**  **программирование** | Программы создаются путем последовательного применения функций, а не инструкций | Haskell, F# | Легкая отладка, Параллельное программирование, Модульность, Отложенное вычисление | плохо подходит для алгоритмов, основанных на графах из-за сравнительно более медленной работы программы | **Pascal**  **Program** z\_20;  **var** a, b, z : real;  **function** f(u, t: real) : real;  **begin**  **if** (u > 0) **and** (t > 0) **then** f := sqr(u) + sqr(t);  **if** (u <= 0) **and** (t <= 0) **then** f := u + sqr(t);  **if** (u > 0) **and** (t <= 0) **then** f := u - t;  **if** (u <= 0) **and** (t > 0) **then** f := u + t;  **end**;  **begin**  a:= 2.5;  b:= -7.3;  z := f(a, b) + f(sqr(a), sqr(b)) + f(sqr(a) - 1, b) + f(a - b, b) + f(sqr(a) + sqr(b), sqr(b) - 1);  write('Z = ', z);  **end**. |
| **8.** | **Логическое**  **программирование** | Подход к программированию, при  котором программа задаѐтся совокупностью правил без явного указания  последовательности их применения. | Prolog, Delta Prolog | Компактность кода, Легкость понимания, Эффективный метод вычислений - рекурсия | Невозможность создания комплексных задач | Python  L = [1, 2, 3, 4, 5]  sum = 0  for x in L:  sum += x |

Для моего проекта я буду использовать [объектно-ориентированное](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [программирование](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) поскольку эта парадигма программирования идеально подходит для создания компьютерных игр, т.к. каждый отдельный элемент в играх представляет собой объект.

**Задание 3. Изучение правил формирования листинга программ.**

import pygame #загрузка стороннего модуля pygame

import os

import config as cfg # загрузка собственного модуля, используемого под именем cfg

pygame.init()

#Класс анимации

class Anime(pygame.sprite.Sprite):#Создание класса анимации

def \_\_init\_\_(self,width,height,path,poverhChego,infinite=False,myText=''):#метод конструктор

pygame.sprite.Sprite.\_\_init\_\_(self)

self.animeMas=[]#создание пустого списка

…………………..

def update(self):# метод, используемый каждый кадр, для изменения обьекта во время игры

……………..

def heroMoveCheck(hero,event):# метод для проверки ввода клавиш движения игроком

if event.type==pygame.KEYDOWN:

if event.key==cfg.KlUp:

hero.movingUp=True

if event.key==cfg.KlDown:

hero.movingDown=True

if event.key==cfg.KlRight:

hero.movingRight=True

if event.key==cfg.KlLeft:

hero.movingLeft=True

if event.key==cfg.KlSpecial:

hero.special()

if event.type==pygame.KEYUP:

if event.key==cfg.KlUp:

hero.movingUp=False

if event.key==cfg.KlDown:

hero.movingDown=False

if event.key==cfg.KlRight:

hero.movingRight=False

if event.key==cfg.KlLeft:

hero.movingLeft=False