

Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт ИМИТ
Кафедра Инфокоммуникаций

Дисциплина Основы кроссплатформенного программирования

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ
Условные операторы и циклы в языке Python3

Выполнил (а):

Клочко Никита

студент 1 курса, 1 группы

специальности ИТС-б-о-21-1

очной формы обучения

Дата защиты

«___» _____ 20__ г.

Оценка _____

Проверил:

Кандидат технических наук,

Доцент кафедры

инфокоммуникаций.

Воронкин Роман Александрович

(Подпись)

Ставрополь, 2021г.

Цель работы: приобретение навыков программирования разветвляющихся алгоритмов и алгоритмов циклической структуры. Освоить операторы языка Python версии 3 `if` , `while` , `for` , `break` и `continue` , позволяющих реализовывать разветвляющиеся алгоритмы и алгоритмы циклической структуры.

Создадим общедоступный репозиторий – <https://github.com/NikitaKloch/laboratornaya-3>

Пример 1:

```
C:\Users\Popa\anaconda3\envs\laba3\python.exe "C:\Users\Popa\anaconda3\envs\laba3\lab1.py"
Connected to pydev debugger (build 203.7148.72)
Value of x? y = -80.58788151475824

Process finished with exit code 0
```

Рисунок 1. Окно вывода для Примера 1.

Пример 2:

```
C:\Users\Popa\anaconda3\envs\laba3\python.exe "C:\Users\Popa\anaconda3\envs\laba3\lab2.py"
Введите номер месяца 8
Summer

Process finished with exit code 0
```

Рисунок 2. Окно вывода для Примера 2.

Пример 3:

```
C:\Users\Popa\anaconda3\envs\laba3\python.exe "C:\Users\Popa\anaconda3\envs\laba3\lab3.py"
Value of n? 6
Value of x? 5
S= 2.896444465760977

Process finished with exit code 0
```

Рисунок 3. Окно вывода для Примера 3.

Пример 4:

```
C:\ALISAGIT\primer3\venv\Scripts\python.exe C:/ALISAGIT/  
Value of a? 52  
x = 7.211102550927979  
X = 7.211102550927978  
  
Process finished with exit code 0
```

Рисунок 4. Окно вывода для Примера 4.

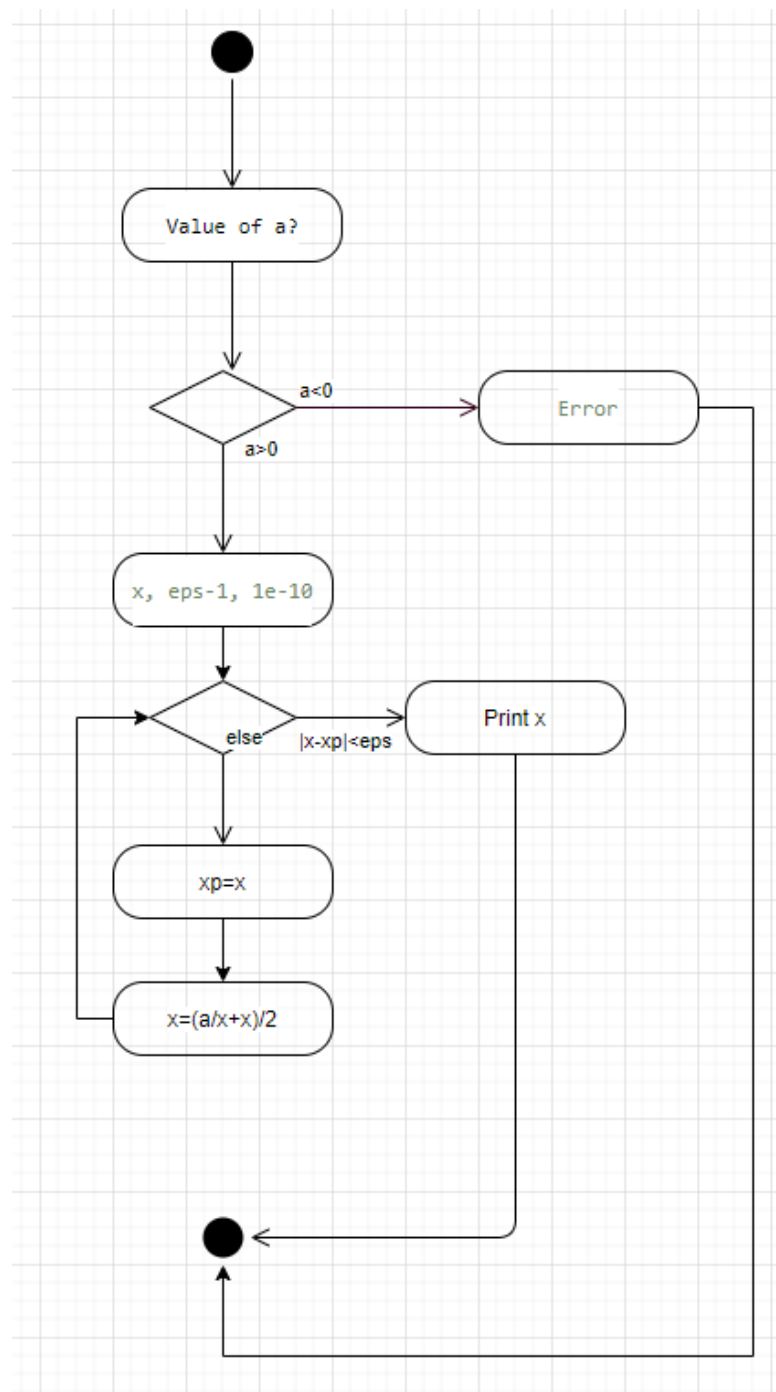


Рисунок 5. UML диаграмма для примера 4.

Пример 5:

```
C:\Users\Popa\anaconda3\envs\laba3  
Value of x? 85  
Ei(85.0) = 9.790723136657533e+34  
  
Process finished with exit code 0
```

Рисунок 6. Окно вывода для Примера 5.

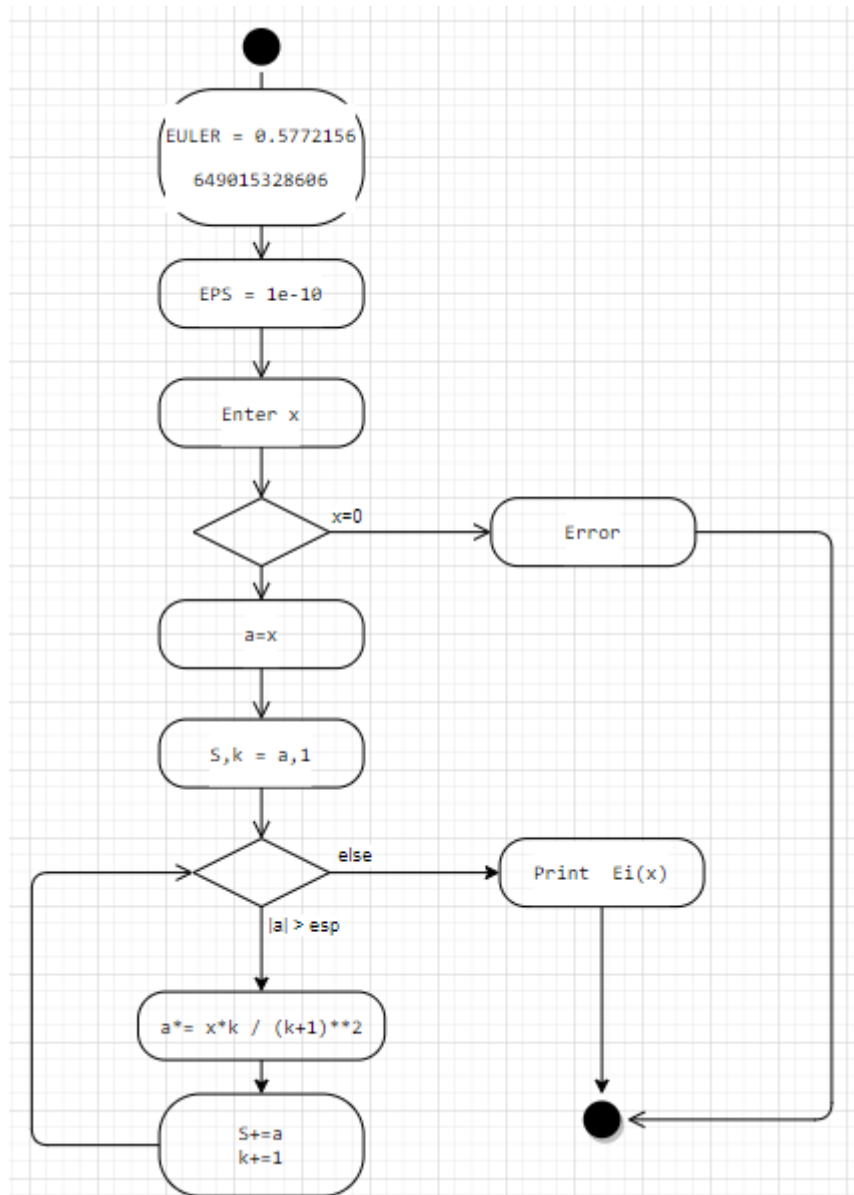


Рисунок 7. UML диаграмма для примера 5.

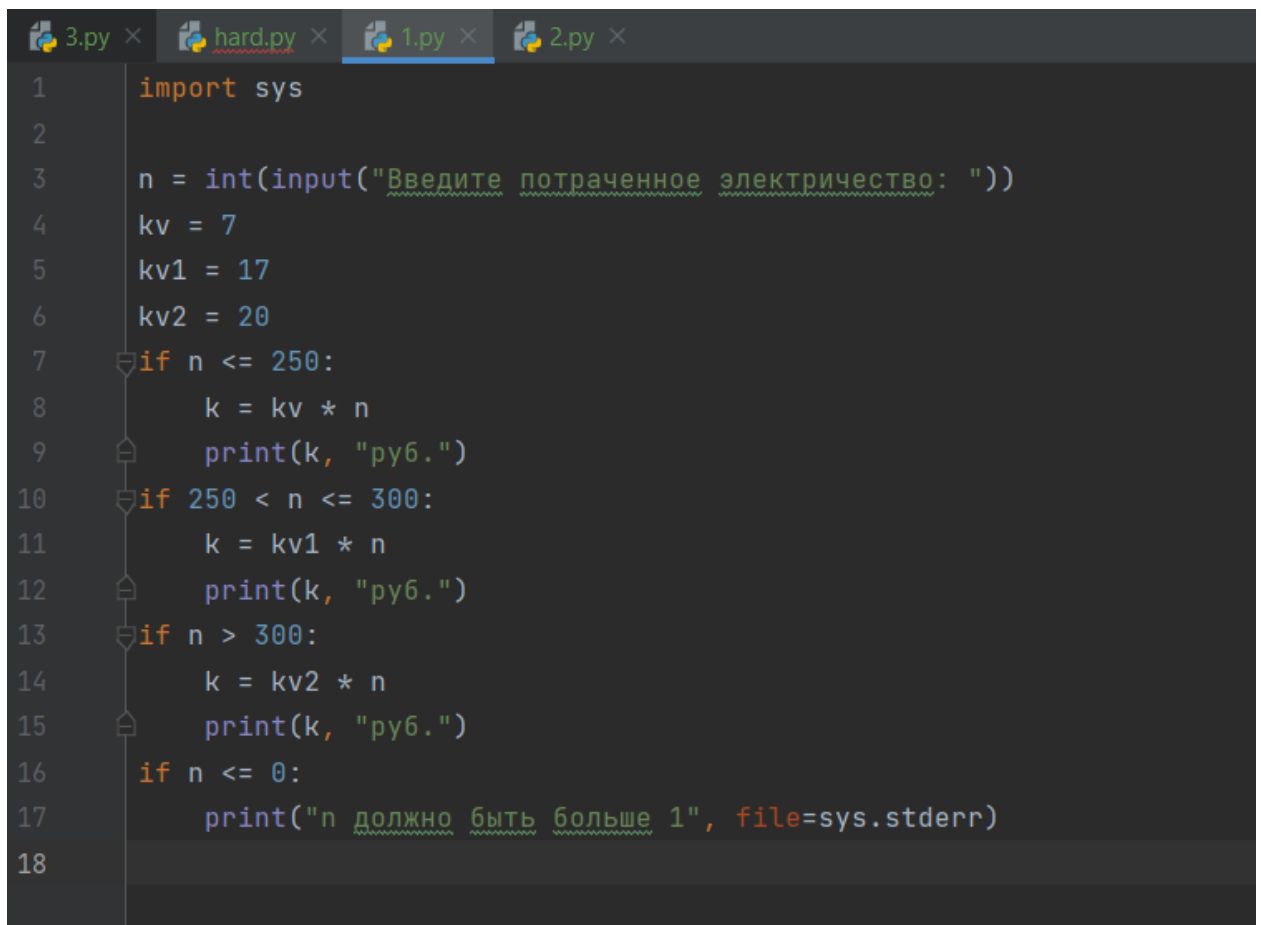
Индивидуальные задания:

Задание 1.

11. Компания по снабжению электроэнергией взимает плату с клиентов по тарифу:

- 7 р. за 1 кВт/ч за первые 250 кВт/ч;
- 17 р. за кВт/ч, если потребление свыше 250, но не превышает 300 кВт/ч;
- 20 р. за кВт/ч, если потребление свыше 300 кВт/ч.

Потребитель израсходовал n кВт/ч. Подсчитать плату.

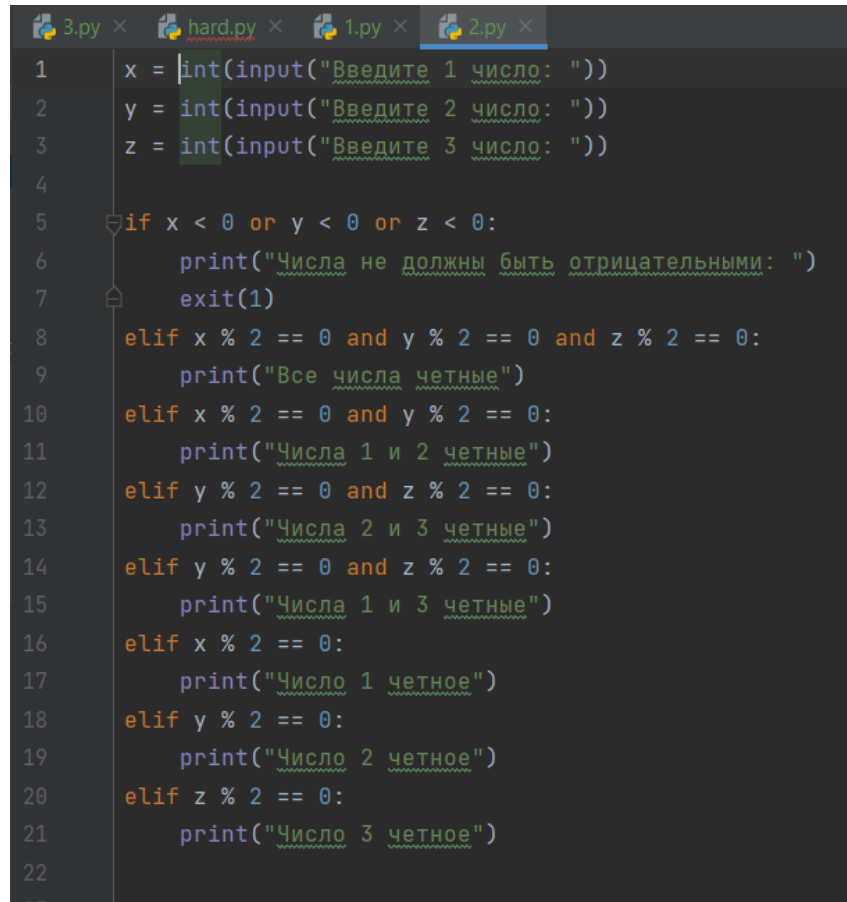


```
1  import sys
2
3  n = int(input("Введите потраченное электричество: "))
4  kv = 7
5  kv1 = 17
6  kv2 = 20
7  if n <= 250:
8      k = kv * n
9      print(k, "руб.")
10 if 250 < n <= 300:
11     k = kv1 * n
12     print(k, "руб.")
13 if n > 300:
14     k = kv2 * n
15     print(k, "руб.")
16 if n <= 0:
17     print("n должно быть больше 1", file=sys.stderr)
18
```

Рисунок 8. Код для Задачи 1.

Задание 2.

11. Определить, есть ли среди трёх заданных чисел чётные.

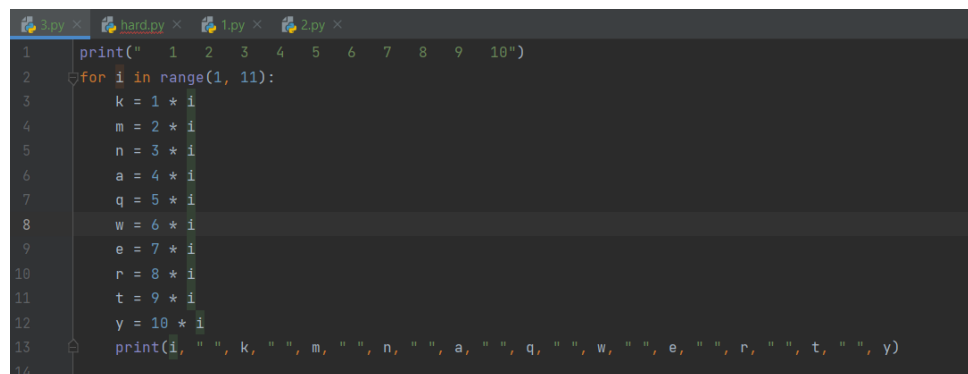


```
1 x = int(input("Введите 1 число: "))
2 y = int(input("Введите 2 число: "))
3 z = int(input("Введите 3 число: "))
4
5 if x < 0 or y < 0 or z < 0:
6     print("Числа не должны быть отрицательными: ")
7     exit(1)
8 elif x % 2 == 0 and y % 2 == 0 and z % 2 == 0:
9     print("Все числа четные")
10 elif x % 2 == 0 and y % 2 == 0:
11     print("Числа 1 и 2 четные")
12 elif y % 2 == 0 and z % 2 == 0:
13     print("Числа 2 и 3 четные")
14 elif y % 2 == 0 and z % 2 == 0:
15     print("Числа 1 и 3 четные")
16 elif x % 2 == 0:
17     print("Число 1 четное")
18 elif y % 2 == 0:
19     print("Число 2 четное")
20 elif z % 2 == 0:
21     print("Число 3 четное")
22
```

Рисунок 9. Код для задачи 2.

Задача 3.

11. Составьте программу, которая печатает таблицу умножения натуральных чисел в десятичной системе счисления.



```
1 print(" 1 2 3 4 5 6 7 8 9 10")
2 for i in range(1, 11):
3     k = 1 * i
4     m = 2 * i
5     n = 3 * i
6     a = 4 * i
7     q = 5 * i
8     w = 6 * i
9     e = 7 * i
10    r = 8 * i
11    t = 9 * i
12    y = 10 * i
13    print(i, " ", k, " ", m, " ", n, " ", a, " ", q, " ", w, " ", e, " ", r, " ", t, " ", y)
14
```

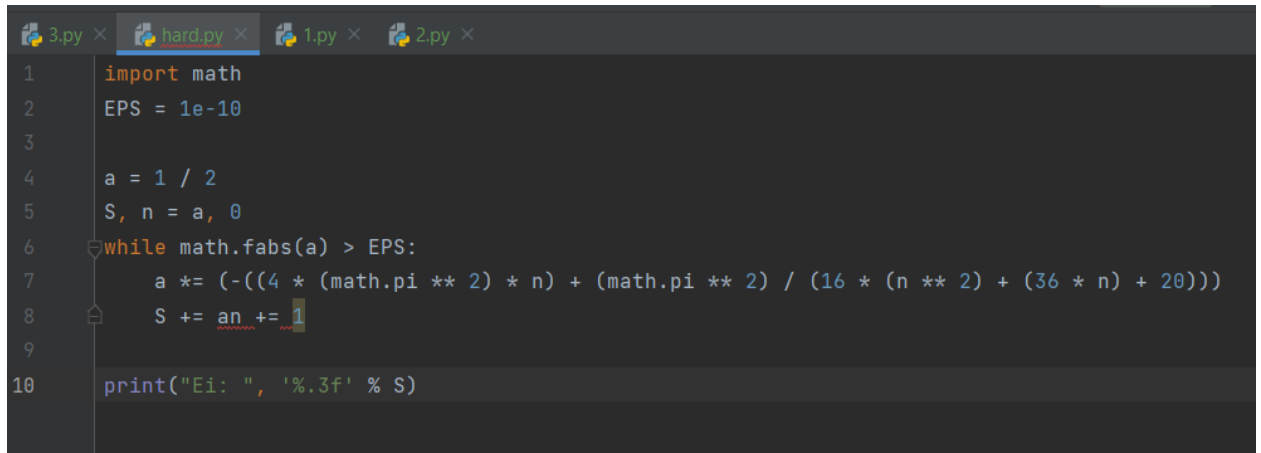
Рисунок 10. Код для задачи 3

Задача повышенной сложности.

5. Первый интеграл Френеля:

$$C(x) = \int_0^x \cos\left(\frac{\pi}{2}t^2\right) dt = \sum_{n=0}^{\infty} \frac{(-1)^n (\pi/2)^{2n}}{(2n)!(4n+1)}. \quad (15)$$

Рисунок 11. Условие задачи повышенной сложности.



```
1 import math
2 EPS = 1e-10
3
4 a = 1 / 2
5 S, n = a, 0
6 while math.fabs(a) > EPS:
7     a *= (-((4 * (math.pi ** 2) * n) + (math.pi ** 2) / (16 * (n ** 2) + (36 * n) + 20)))
8     S += a * n += 1
9
10 print("Ei: ", '%.3f' % S)
```

Рисунок 12. Код задачи повышенной сложности.

Ответы на контрольные вопросы:

1. Для чего нужны диаграммы деятельности UML?

Диаграммы деятельности. Унифицированный язык моделирования (UML) является стандартным инструментом для создания «чертежей» программного обеспечения. С помощью UML можно визуализировать, специфицировать, конструировать и документировать артефакты программных систем. UML пригоден для моделирования любых систем: от информационных систем масштаба предприятия до распределенных Web-приложений и даже встроенных систем реального времени. Это очень выразительный язык, позволяющий рассмотреть систему со всех точек зрения, имеющих отношение к ее разработке и последующему развертыванию. Несмотря на обилие выразительных возможностей, этот язык прост для понимания и использования.

Диаграммы деятельности - это один из пяти видов диаграмм, применяемых в UML для моделирования динамических аспектов поведения системы. Диаграмма деятельности - это, по существу, блок-схема, которая показывает, как поток управления переходит от одной деятельности к другой, однако, по сравнению с последней, у ней есть явные преимущества: поддержка многопоточности и объектно-ориентированного проектирования.

Диаграмма деятельности (Activity diagram) показывает поток переходов от одной деятельности к другой. Деятельность (Activity) - это продолжающийся во времени неатомарный шаг вычислений в автомате. Деятельности в конечном счете приводят к выполнению некоего действия (Action), составленного из выполняемых атомарных вычислений, каждое из которых либо изменяет состояние системы, либо возвращает какое-то значение. Действие может заключаться в вызове другой операции, отправке сигнала, создании или уничтожении объекта либо в простом вычислении - скажем, значения выражения. Графически диаграмма деятельности представляется в виде графа, имеющего вершины и ребра.

2. Что такое состояние действия и состояние деятельности?

В потоке управления, моделируемом диаграммой деятельности, происходят различные события. Вы можете вычислить выражение, в результате чего изменяется значение некоторого атрибута или возвращается некоторое значение. Также, например, можно выполнить операцию над объектом, послать ему сигнал или даже создать его или уничтожить. Все эти выполняемые атомарные вычисления называются состояниями действия, поскольку каждое из них есть состояние системы, представляющее собой выполнение некоторого действия. Состояния действия изображаются прямоугольниками с закругленными краями. Внутри такого символа можно записывать произвольное выражение.

Состояния действия не могут быть подвергнуты декомпозиции. Кроме того, они атомарны. Это значит, что внутри них могут происходить различные события, но выполняемая в состоянии действия работа не может быть прервана. Обычно предполагается, что длительность одного состояния действия занимает неощутимо малое время.

В противоположность этому состояния деятельности могут быть подвергнуты дальнейшей декомпозиции, вследствие чего выполняемую деятельность можно представить с помощью других диаграмм деятельности. Состояния деятельности не являются атомарными, то есть могут быть прерваны. Предполагается, что для их завершения требуется заметное время. Можно считать, что состояние действия - это частный вид состояния деятельности, а конкретнее – такое состояние, которое не может быть подвергнуто дальнейшей декомпозиции. А состояние деятельности можно представлять себе как составное состояние, поток управления которого включает только другие состояния деятельности и действий.

3. Какие нотации существуют для обозначения переходов и ветвлений в диаграммах деятельности?

Переходы. Когда действие или деятельность в некотором состоянии завершается, поток управления сразу переходит в следующее состояние действия или деятельности. Для описания этого потока используются

переходы, показывающие путь из одного состояния действия или деятельности в другое. Поток управления должен где-то начинаться и заканчиваться (разумеется, если это не бесконечный поток, у которого есть начало, но нет конца). Как показано на рисунке, вы можете задать как начальное состояние (закрашенный кружок), так и конечное (закрашенный кружок внутри окружности).

Ветвление. Простые последовательные переходы встречаются наиболее часто, но их одних недостаточно для моделирования любого потока управления. Как и в блок-схеме, вы можете включить в модель ветвление, которое описывает различные пути выполнения в зависимости от значения некоторого булевского выражения. Точка ветвления представляется ромбом. В точку ветвления может входить ровно один переход, а выходить – два или более. Для каждого исходящего перехода задается булевское выражение, которое вычисляется только один раз при входе в точку ветвления. Ни для каких двух исходящих переходов эти сторожевые условия не должны одновременно принимать значение «истина», иначе поток управления окажется неоднозначным. Но эти условия должны покрывать все возможные варианты, иначе поток остановится.

Для удобства разрешается использовать ключевое слово `else` для пометки того из исходящих переходов, который должен быть выбран в случае, если условия, заданные для всех остальных переходов, не выполнены.

Реализовать итерацию можно, если ввести два состояния действия - в первом устанавливается значение счетчика, во втором оно увеличивается - и точку ветвления, вычисление в которой показывает, следует ли прекратить итерации.

4. Какой алгоритм является алгоритмом разветвляющейся структуры?

Алгоритм разветвляющейся структуры - это алгоритм, в котором вычислительный процесс осуществляется либо по одной, либо по другой ветви, в зависимости от выполнения некоторого условия. Программа

разветвляющейся структуры реализует такой алгоритм. В программе разветвляющейся структуры имеется один или несколько условных операторов. Для программной реализации условия используется логическое выражение. В сложных структурах с большим числом ветвей применяют оператор выбора.

5. Чем отличается разветвляющийся алгоритм от линейного?

Линейный алгоритм - алгоритм, все этапы которого выполняются однократно и строго последовательно.

Разветвляющийся алгоритм - алгоритм, содержащий хотя бы одно условие, в результате проверки которого ЭВМ обеспечивает переход на один из двух возможных шагов.

6. Что такое условный оператор? Какие существуют его формы?

Оператор ветвления `if` позволяет выполнить определенный набор инструкций в зависимости от некоторого условия. Возможны следующие варианты использования.

Синтаксис оператора `if` выглядит так. После оператора `if` записывается выражение. Если это выражение истинно, то выполняются инструкции, определяемые данным оператором. Выражение является истинным, если его результатом является число не равное нулю, непустой объект, либо логическое `True`. После выражения нужно поставить двоеточие `“:”`.

Бывают случаи, когда необходимо предусмотреть альтернативный вариант выполнения программы. Т. е. при истинном условии нужно выполнить один набор инструкций, при ложном – другой. Для этого используется конструкция `if – else`.

Оператор цикла `while` выполняет указанный набор инструкций до тех пор, пока условие цикла истинно. Истинность условия определяется также как и в операторе `if`.

При работе с циклами используются операторы `break` и `continue`.

Оператор `break` предназначен для досрочного прерывания работы цикла `while`.

Оператор continue запускает цикл заново, при этом код, расположенный после данного оператора, не выполняется.

Оператор for выполняет указанный набор инструкций заданное количество раз, которое определяется количеством элементов в наборе.

Функция range возвращает неизменяемую последовательность чисел в виде объекта range.

Параметры функции:

Start - с какого числа начинается последовательность. По умолчанию - 0

Stop - до какого числа продолжается последовательность чисел.

Указанное число не включается в диапазон

Step - с каким шагом растут числа. По умолчанию 1

Функция range хранит только информацию о значениях start, stop и step и вычисляет значения по мере необходимости. Это значит, что независимо от размера диапазона, который описывает функция range, она всегда будет занимать фиксированный объем памяти.

7. Какие операторы сравнения используются в Python?

В Python есть шесть операций сравнения. Все они имеют одинаковый приоритет, который выше, чем у логических операций.

Разрешенные операции сравнения:

- $x < y$ – строго x меньше y ,
- $x \leq y$ – x меньше или равно y ,
- $x > y$ – строго x больше y ,
- $x \geq y$ – x больше или равно y ,
- $x == y$ – x равно y ,
- $x != y$ – x не равно y .

8. Что называется простым условием? Приведите примеры.

Простым условием (отношением) **называется** выражение, составленное из двух арифметических выражений или двух текстовых величин (иначе их еще **называют** операндами), связанных одним из знаков: <

- меньше, чем... Например, простыми отношениями являются следующие: $x < y$; $k \leq \sqrt{c} + \text{abs}(a+b)$; $9 < 11$; 'мама' <> 'папа'.

9. Что такое составное условие? Приведите примеры.

Составные условия — это условия, состоящие из двух или более простых условий, соединенных с помощью логических операций: `and`, `or`, `not`. Простые условия при этом заключаются в скобки.

Примеры составных условий:

`(a < 5) and (b > 8)`

`(x >= 0) or (x < -3)`

`not (a = 0) or (b = 0)`

10. Какие логические операторы допускаются при составлении сложных условий?

Используются специальные операторы, объединяющие два и более простых логических выражения. Широко используются два оператора – так называемые логические И (**and**) и ИЛИ (**or**).

11. Может ли оператор ветвления содержать внутри себя другие ветвления?

Да. Такой оператор ветвления называется вложенным. В этом случае важно не перепутать какая ветвь кода к какому оператору относится. Поэтому рекомендуется соблюдать отступы в исходном коде программы, чтобы не запутаться. `if <условие> then if< условие> then< оператор 1>;` Вложенный условный оператор.

12. Какой алгоритм является алгоритмом циклической структуры?

Например,

Перевод текста с иностранного языка (прочитать первое предложение, перевести, записать и т.д.)

Построение графика функции по точкам (взять первый аргумент, вычислить значение функции, построить точку и т.д.)

13. Типы циклов в языке Python.

Цикл — это блок, элементы которого повторяют своё действие заданное количество раз. Бывают. Практически все современные алгоритмы содержат в себе **циклы**. В языке **Python** существуют следующие **типы циклов**: 1. **While()** — **Цикл** с предусловием 2. **for()** — **Цикл** с чётким количеством проходов.

14. Назовите назначение и способы применения функции **range**.

Функция range является одной из встроенных **функций**, доступных в Python. Он генерирует серию целых чисел, от значения **start** до **stop**, **указанного** пользователем. Мы можем использовать его для цикла **for** и обходить весь диапазон как список. **Функция range ()** принимает один обязательный и два необязательных параметра. Это работает по-разному с различными комбинациями аргументов.

15. Как с помощью функции **range** организовать перебор значений от 15 до 0 с шагом 2?

```
range(15, 0, -2)
```

16. Могут ли циклы быть вложенными?

Чисто технически во **вложенных циклах** нет ничего особенного. Их можно **вкладывать** внутрь любого блока и друг в друга сколько угодно раз. Но прямой связи между внешним и **вложенным циклами** нет. Внутренний **цикл** может использовать результаты внешнего, а может и работать по своей собственной логике независимо. **Вложенные циклы** коварны. Их наличие может резко увеличить сложность кода, так как появляется множество постоянно изменяющихся переменных.

17. Как образуется бесконечный цикл и как выйти из него?

Бесконечный цикл в программировании — цикл, написанный таким образом, что условие выхода из него никогда не выполняется. О программе, вошедшей в бесконечный цикл, иногда говорят, что она *зациклилась*.

18. Для чего нужен оператор **break**?

Выражение **break** заставляет программу выйти из цикла.

19. Где употребляется оператор **continue** и для чего он используется?

Выражение Continue. Выражение continue дает возможность пропустить часть цикла, где активируется внешнее условие, но при этом выполнить остальную часть цикла. При этом прерывается текущая итерация цикла, но программа возвращается к началу цикла. Выражение continue размещается в блоке кода под выражением цикла, обычно после условного выражения if.

20. Для чего нужны стандартные потоки stdout и stderr?

STDOUT - стандартный вывод - то, куда выводят данные команды echo/print, консоль или сокет отправляющий данные браузеру. **STDERR** – поток сообщений об ошибках.

21. Как в Python организовать вывод в стандартный поток stderr?

22. Каково назначение функции exit?

Функция exit() вызывает немедленное нормальное завершение программы.

Вывод: Приобрел навыки программирования разветвляющихся алгоритмов и алгоритмов циклической структуры. Освоил операторы языка Python.3 if, while, for, break, continue, позволяющих реализовывать разветвляющиеся алгоритмы и алгоритмы циклической структуры.