МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования «Северо-Кавказский федеральный университет»

Кафедра инфокоммуникаций

Отчет по лабораторной работе №5

по дисциплине «Основы программной инженерии»

Выполнил студент группы ПИЖ-б-о-20-1
Ваньянц И.М. « »_____20__г.
Подпись студента_____
Работа защищена « »_____20__г.
Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2021

ХОД РАБОТЫ

1. Пример 1

```
import math

if __name__ == '__main__':
    x =float(input("Value of x? "))

    if x<= 0:
        y=2*x*x + math.cos(x)
    elif x<5:
        y = x+1
    else:
        y = math.sin(x) - x*x

    print(f"y = {y}")
```

Рисунок 1 – Код примера 1

```
Value of x? -5
y = 50.28366218546323
```

Рисунок 2 – Пример работы программы для x<=0

```
Value of x? 4
y = 5.0
```

Рисунок 3 – Пример работы программы для x<5

```
Value of x? 35
y = -1225.4281826694962
```

Рисунок 4 – Пример работы программы при else

2. Пример 2

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys

if __name__ == '__main__':
    n = int(input("Введите номер месяца: "))

    if n == 1 or n == 2 or n == 12:
        print("Зима")
    elif n == 3 or n == 4 or n == 5:
        print("Весна")
    elif n == 6 or n == 7 or n == 8:
        print("Лето")
    elif n == 9 or n == 10 or n == 11:
        print("Осень")
    else:
        print("Ошибка!", file=sys.stderr)
        exit(1)
```

Рисунок 5 – код программы

```
Введите номер месяца: 1
Зима

Process finished with exit code 0
```

Рисунок 6 – пример работы программы при n=1

```
Введите номер месяца: 3
Весна

Process finished with exit code 0
```

Рисунок 7 – пример работы программы при n=3

```
Введите номер месяца: 6
Лето

Process finished with exit code 0
```

Рисунок 8 – пример работы программы при n=6

```
Введите номер месяца: 9
Осень

Process finished with exit code 0
```

Рисунок 9 – пример работы программы при n=9

```
Введите номер месяца: 13
Ошибка!

Process finished with exit code 1
```

Рисунок 10 – пример работы программы при n=13

3. Пример 3

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import math

if __name__ == '__main__':
    n = int(input("Value of n? "))
    x = float(input("Value of x? "))
    S = 0.0
    for k in range(1, n + 1):
        a = math.log(k * x) / (k * k)
        S += a

    print(f"S = {S}")
```

рисунок 11 – код программы

```
Value of n? 4
Value of x? 5
S = 2.6732119195688706

Process finished with exit code 0
```

рисунок 12 – пример работы программы при n=4, n=5

4. Пример 4

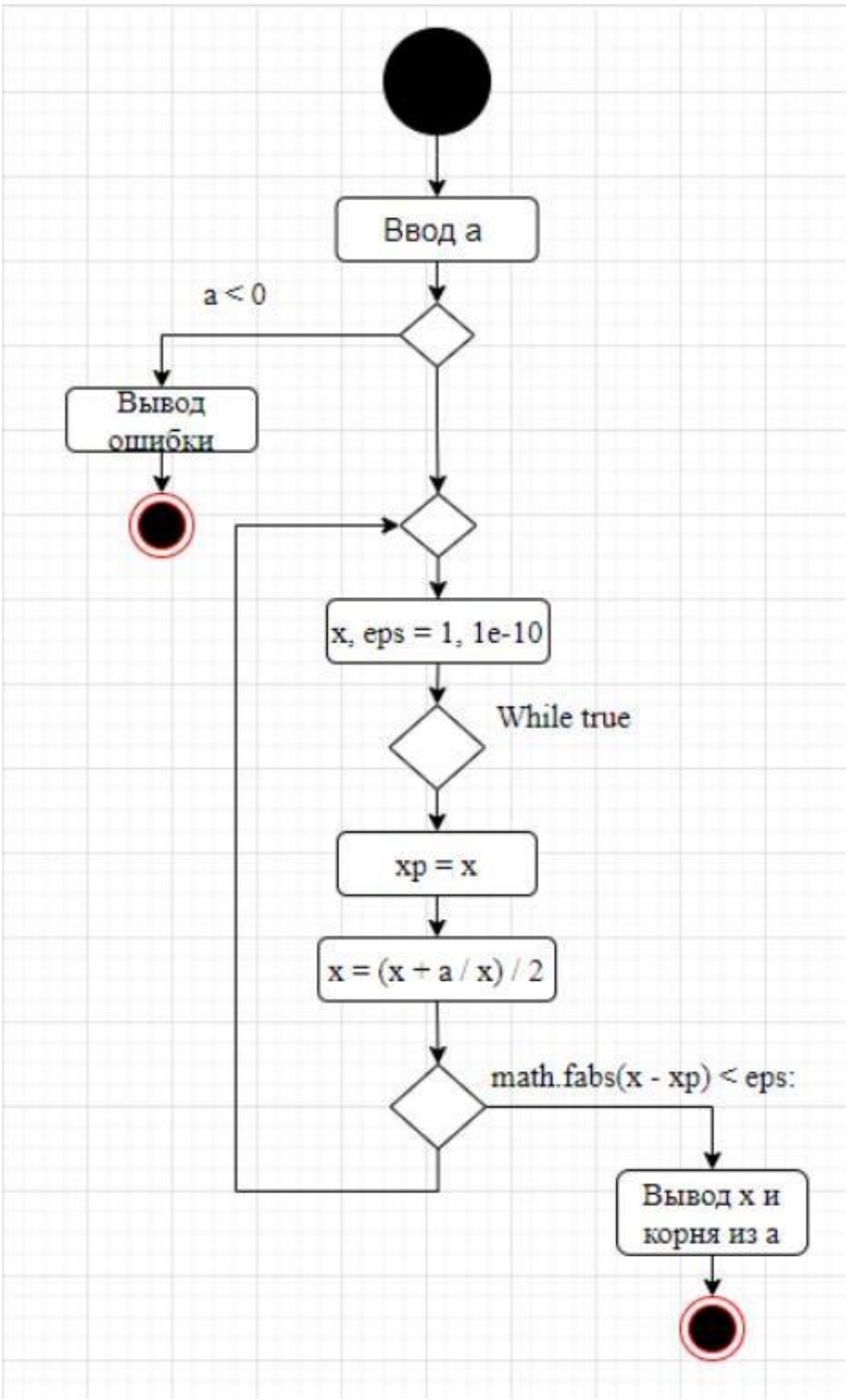

Рисунок 13 – UML-диаграмма алгоритма

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import math
import sys

if __name__ == '__main__':
    a = float(input("Value of a? "))
    if a < 0:
        print("Illegal value of a", file=sys.stderr)
        exit(1)
    x, eps = 1, 1e-10
    while True:
        xp = x
        x = (x + a / x) / 2
        if math.fabs(x - xp) < eps:
            break

    print(f"x = {x}\nX = {math.sqrt(a)}")
```

Рисунок 14 – код программы

```
Value of a? -1
Illegal value of a

Process finished with exit code 1
```

рисунок 15 – пример работы программы при a<0

```
Value of a? 4
x = 2.0
X = 2.0

Process finished with exit code 0
```

рисунок 16 – пример работы программы при a=4

5. Пример 5

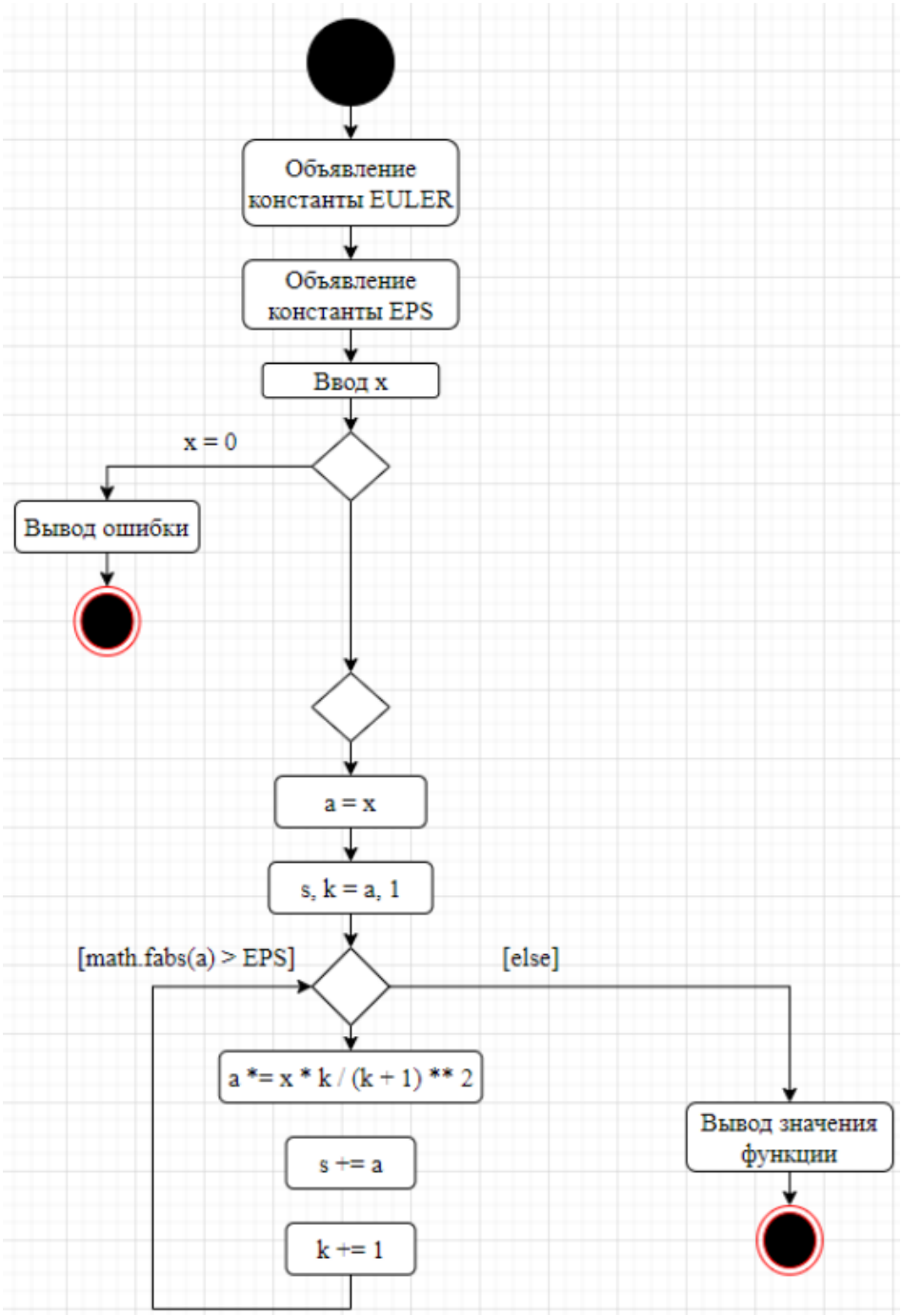

Рисунок 17 – UML-диаграмма алгоритма

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import math
import sys

# Постоянная Эйлера.
EULER = 0.5772156649015328606
# Точность вычислений.
EPS = 1e-10

if __name__ == '__main__':
    x = float(input("Value of x? "))
    if x == 0:
        print("Illegal value of x", file=sys.stderr)
        exit(1)
    a = x
    S, k = a, 1

    # Найти сумму членов ряда.
    while math.fabs(a) > EPS:
        a *= x * k / (k + 1) ** 2
        S += a
        k += 1

    # Вывести значение функции.
    print(f"Ei({x}) = {EULER + math.log(math.fabs(x)) + S}")
```

Рисунок 18 – код программы

```
Value of x? 0
Illegal value of x

Process finished with exit code 1
```

Рисунок 19 – пример работы программы при x=0

```
Value of x? 4
Ei(4.0) = 19.63087447005282

Process finished with exit code 0
```

Рисунок 20 – пример работы программы при x=4

6. Индивидуальное задание 1

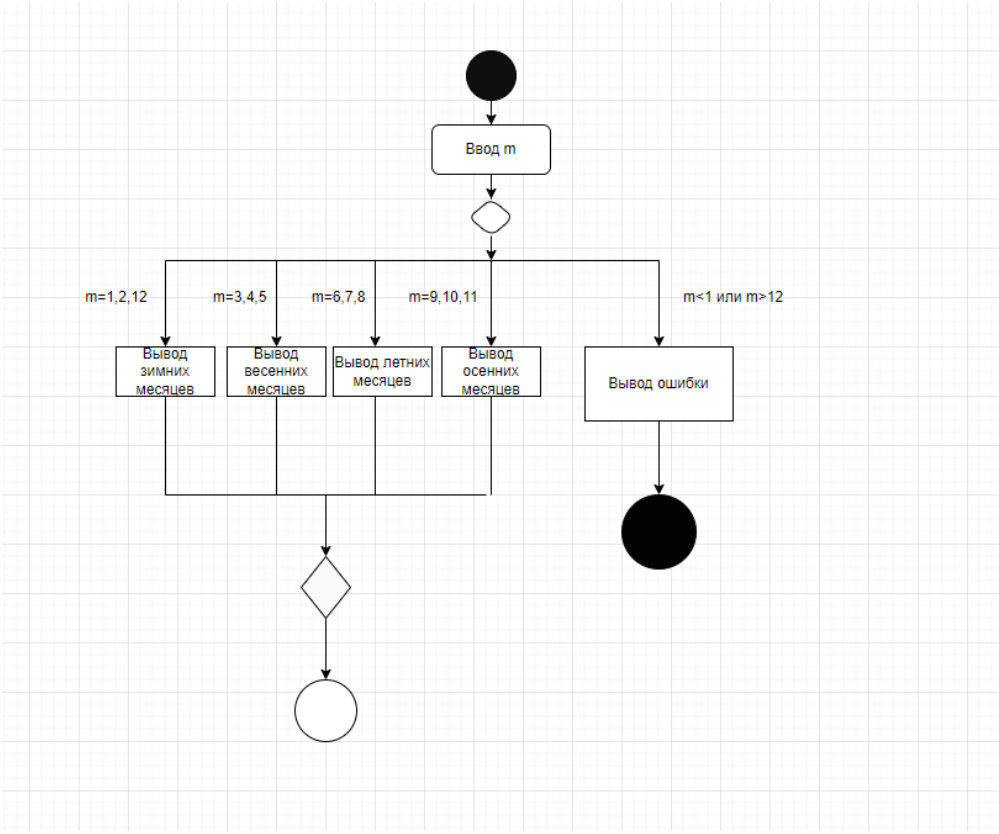

Рисунок 21 – UML диаграмма

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys

if __name__ == '__main__':
    m = int(input("Введите номер месяца: "))

    if m == 1 or m == 2 or m == 12:
        print("Декабрь, Январь, Февраль")
    elif m == 3 or m == 4 or m == 5:
        print("Март, Апрель, Май")
    elif m == 6 or m == 7 or m == 8:
        print("Июнь, Июль, Август")
    elif m == 9 or m == 10 or m == 11:
        print("Сентябрь, Октябрь, Ноябрь")
    else:
        print("Ошибка!", file=sys.stderr)
        exit(1)
```

Рисунок 22 - код программы

```
Введите номер месяца: 1
Декабрь, Январь, Февраль
```

Рисунок 23 – результат при вводе 1

7. Индивидуальное задание 2

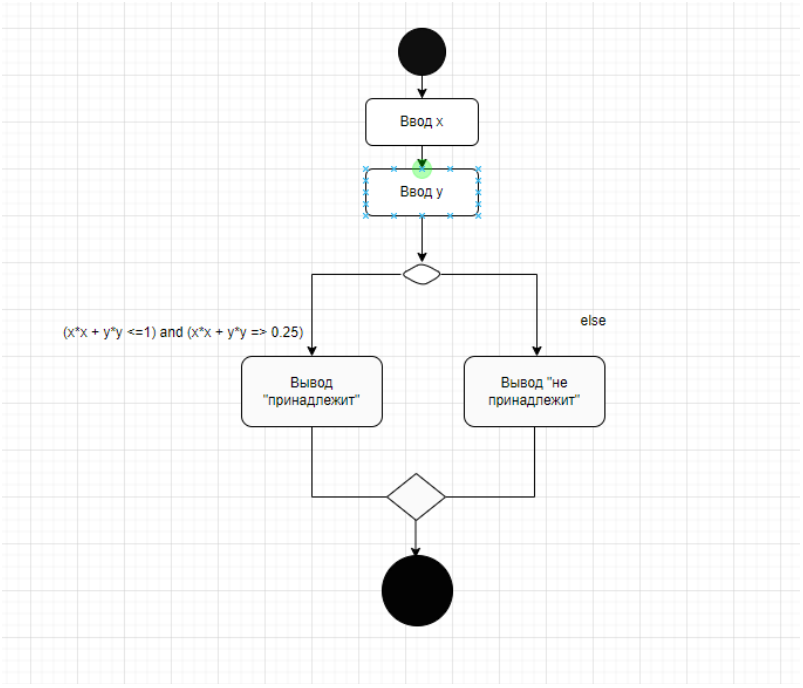

Рисунок 24 – UML - диаграмма

```python
x = float(input("Введите координату x: "))
y = float(input("Введите координату y: "))
if (x*x + y*y <=1) and (x*x + y*y >= 0.25):
    print("Принадлежит")
else:
    print("Не принадлежит")
```

Рисунок 23 – код программы

```
Введите координату x: 20
Введите координату y: 20
Не принадлежит
```

Рисунок 24 – результат программы “не принадлежит”

```
Введите координату x: 0.5
Введите координату y: 0.5
Принадлежит
```

Рисунок 25 – результат программы “принадлежит”

8. Индивидуальное задание 3

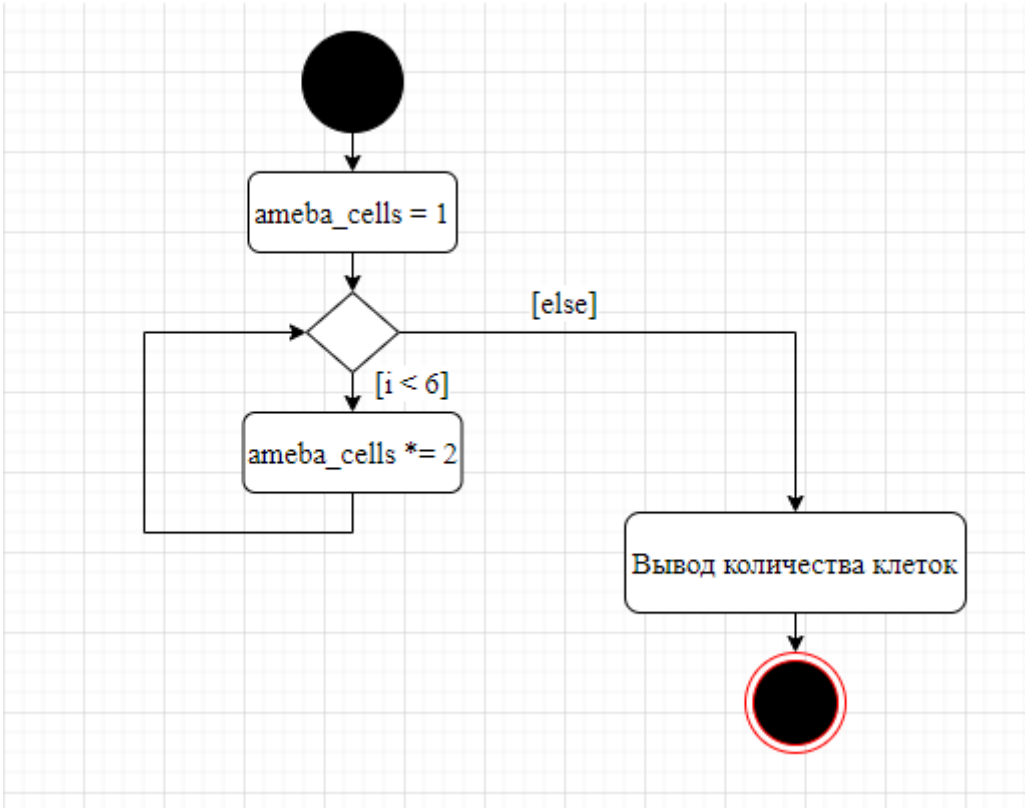

Рисунок 26 – UML диаграмма

```
ameba_cells = 1
for i in range(6):
    ameba_cells *= 2
print(f"Будет {ameba_cells} клеток через 6 часов ")
```

Рисунок 27 – код программы

```
Будет 64 клетки через 6 часов

Process finished with exit code 0
```

Рисунок 28 – результат программы

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

1. 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.

2. 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 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. В UML переход представляется простой линией со стрелкой. Точка ветвления представляется ромбом. В точку ветвления может входить ровно один переход, а выходить – два или более.
4. Алгоритм разветвляющейся структуры - это алгоритм, в котором вычислительный процесс осуществляется либо по одной, либо по другой ветви, в зависимости от выполнения некоторого условия.
5. Линейный алгоритм выполняется последовательно независимо от чего-либо, а алгоритм ветвления выполняется определенные действия в зависимости от выполнения условия или условий.
6. Оператор ветвления «if» позволяет выполнить определенный набор инструкций в зависимости от некоторого условия. Возможны следующие варианты использования: 1) Конструкция «if» 2) Конструкция «if» - «else» 3) Конструкция «if» - «elif» - «else»
7. , <=, >=, ==, !=.
8. Логические выражения являются простыми, если в них выполняется только одна логическая операция. «x > 15» или «a != b»
9. В составных условиях используется 2 и более логические операции. «x > 8 and y <= 3»
10. and и or
11. Да, может.
12. Алгоритм циклической структуры - это алгоритм, в котором происходит многократное повторение одного и того же участка программы. Такие повторяемые участки вычислительного процесса называются циклами.
13. Цикл «while» и цикл «for».
14. 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.
15. range(15, 0, -2).
16. Да, могут.
17. Пример бесконечного цикла: a = 0 while a >= 0: if a == 7: break a += 1 print("A") Оператор «break» предназначен для досрочного прерывания работы цикла «while».
18. Оператор «break» предназначен для досрочного прерывания работы цикла «while».
19. Оператор «continue» запускает цикл заново, при этом код, расположенный после данного оператора, не выполняется.
20. В операционной системе по умолчанию присутствуют стандартные потоки вывода на консоль: буферизованный поток stdout для вывода данных и информационных сообщений, а также небуферизованный поток stderr для вывода сообщений об ошибках. По умолчанию функция print использует поток stdout. Хорошим стилем программирования является наличие вывода ошибок в стандартный поток stderr поскольку вывод в потоки stdout и stderr может обрабатываться как операционной системой, так и сценариями пользователя по-разному.

21. Для того, чтобы использовать поток stderr необходимо передать его в параметре file функции print. 22. В Python завершить программу и передать операционной системе заданныq код возврата можно посредством функции exit. В данном примере выполняется вызов exit(1) , что приводит к немедленному завершению программы и операционной системе передается 1 в качестве кода возврата, что говорит о том, что в процессе выполнения программы произошли ошибки.