

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ ФГАОУ ВО  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития  
Кафедра инфокоммуникаций

Отчет по лабораторной работе №3

По дисциплине основы кроссплатформенного программирования

«Условные операторы и циклы в языке Python»

Выполнила:

студентк группы ИТС-б-о-21-1

Абдикодиров Жахонгир Хуснитдин  
угли

---

(подпись)

Проверил: Доцент, к.т.н, доцент  
кафедры

инфокоммуникаций

Воронкин Р. А.

Работа защищена с оценкой:

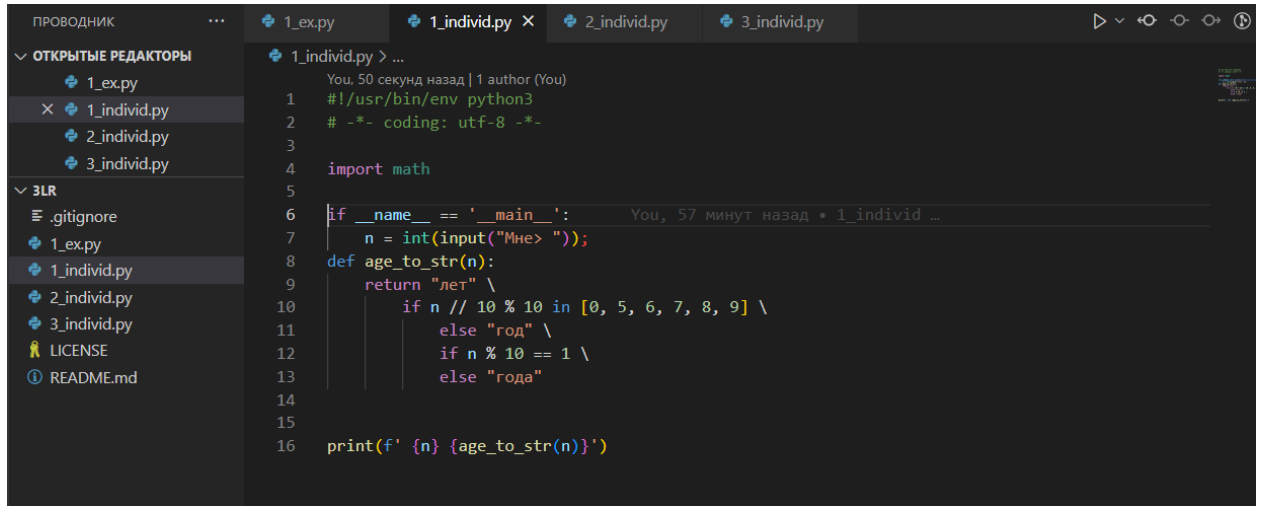
---

(подпись)

Ставрополь, 2022

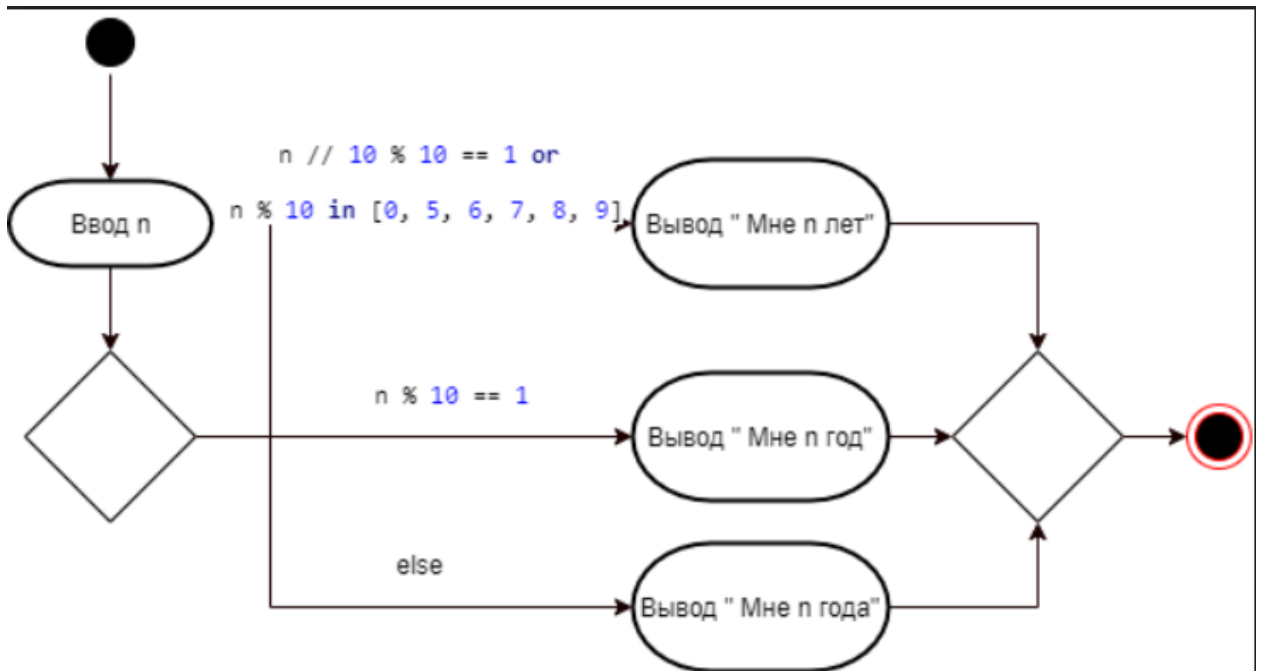
## Индивидуальные задания:

1 индивидуальное задание код и UML-диаграмма 1 вариант:



The screenshot shows a code editor with four tabs: 1\_ex.py, 1\_individ.py (active), 2\_individ.py, and 3\_individ.py. The active tab contains the following Python code:

```
1  You, 50 секунд назад | 1 author (You)
2  1  #!/usr/bin/env python3
3  2  # -*- coding: utf-8 -*-
4  3
5  4  import math
6  5
7  6  if __name__ == '__main__':
8  7      n = int(input("Мне> "));
9  8  def age_to_str(n):
10  9      return "лет" \
11 10         if n // 10 % 10 in [0, 5, 6, 7, 8, 9] \
12 11         else "год" \
13 12         if n % 10 == 1 \
14 13         else "года"
15 14
16 15 print(f' {n} {age_to_str(n)}')
```



## 2 индивидуальное задание код 1 вариант:

```
2_individ.py > ...
You, 1 минуту назад | 1 author (You)
1  import random
2
3  if __name__ == '__main__':
4      title = {
5          1 : 'крысы',
6          2 : 'коровы',
7          3 : 'тигра',
8          4 : 'зайца',
9          5 : 'дракона',
10         6 : 'змеи',
11         7 : 'лошади',
12         8 : 'овцы',
13         9 : 'обезьяны',
14         10 : 'курицы',
15         11 : 'собаки',
16         12 : 'свиньи'
17     }
18     color = {
19         0 : {
20             1 : 'зеленой',
21             2 : 'красной',
22             3 : 'желтой',
23             4 : 'белой',
24             5 : 'черной'
25         },
26         1 : {
27             1 : 'зеленого',
28             2 : 'красного',
29             3 : 'желтого',
30             4 : 'белого',
31             5 : 'черного'
32         }
33     }
34     N = random.randrange(1900,2222)
35     N = 1984
36     N = 2012
37     print("Год:",N)
38     year_name = 'Год '
39     title_code = (N - 4) % 12 + 1
40     print("Title code: ", title_code)
41     print("Title: ", title[title_code])
42     i = 0
43     if title_code in [3,4,5]:
44         i = 1
45     color_code = (N - 4) % 10 + 1
46     color_code = int((color_code - 1) / 2) + 1
47     print("Color code: ", color_code)
48     print("Color: ", color[i][color_code])
49     year_name += color[i][color_code] + ' ' + title[title_code]
50     print(year_name)
```

### 3 индивидуальное задание код и UML-диаграмма 1 вариант:

ОТКРЫТЫЕ РЕДАКТОРЫ

1\_ex.py

1\_individ.py

2\_individ.py

3\_individ.py

3LR

.gitignore

1\_ex.py

1\_individ.py

2\_individ.py

3\_individ.py

LICENSE

README.md

СТРУКТУРА

ВРЕМЕННАЯ ШКАЛА

3\_individ.py > ...

You, 15 секунд назад | 1 author (You)

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  if __name__ == '__main__':
5      for i in range(1, 10):
6          for j in range(8, 10):
7              for k in range(0, 10):
8                  if i + j + k == i * j * k:
9                      print(f"{i}{j}{k}")
10
11
12
13
```

You, 1 секунду назад • Uncommitted

ПРОБЛЕМЫ

ВЫХОДНЫЕ ДАННЫЕ

КОНСОЛЬ ОТЛАДКИ

ТЕРМИНАЛ

GITLENS

PS C:\Users\Admin\Desktop\3LR\3LR> git push

Enumerating objects: 5, done.

Counting objects: 100% (5/5), done.

Delta compression using up to 4 threads

Compressing objects: 100% (3/3), done.

Writing objects: 100% (3/3), 420 bytes | 420.00 KiB/s, done.

Total 3 (delta 1), reused 0 (delta 0), pack-reused 0

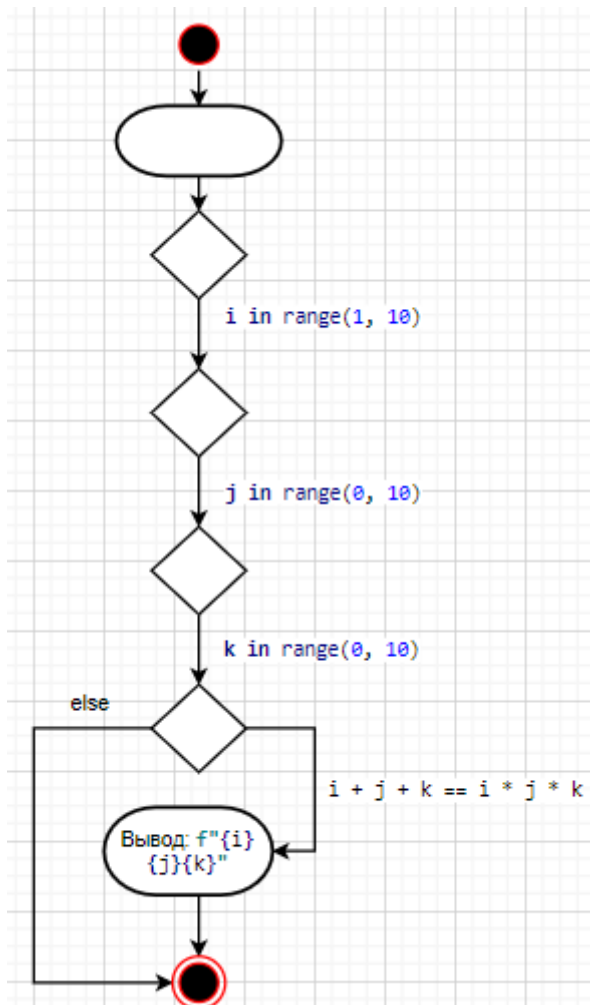
remote: Resolving deltas: 100% (1/1), completed with 1 local object.

To https://github.com/dzsesakq/3LR.git

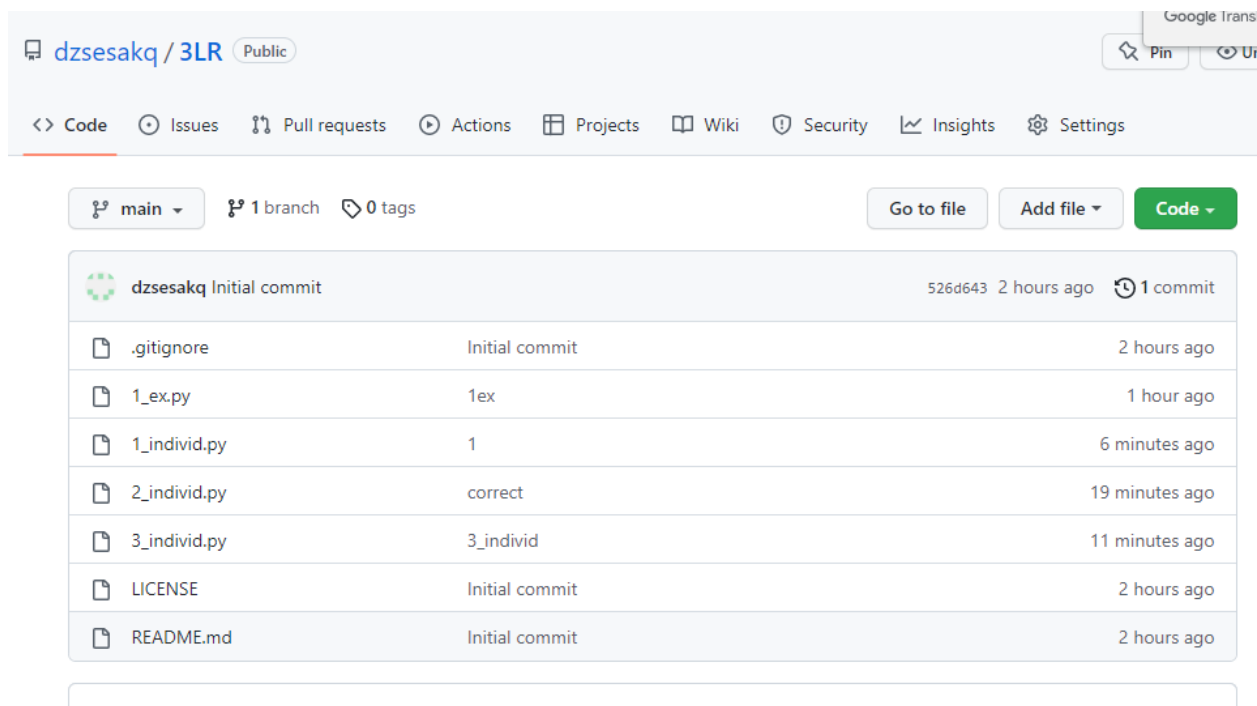
69c0ba5..3aa1568 main -> main

PS C:\Users\Admin\Desktop\3LR\3LR>

Активация  
Чтобы активир  
раздел "Парам



Все сделанные изменения отправлены на сервер GitHub.



Вопросы для защиты работы:

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, послыке 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?

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. Как показано на рис. 4.1, 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. В UML переход представляется простой линией со стрелкой, как показано на рис. 4.2.

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. Как видно из рис. 4.3, 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.

### 3. 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.

### 4. 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.

Разница между ними состоит в том, что тело цикла с постусловием всегда выполняется хотя бы один раз, после чего проверяется, надо ли его выполнять еще раз. Проверка необходимости выполнения цикла с предусловием делается до тела цикла, поэтому возможно, что он не выполнится ни разу.

### 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.

Конструкция `if`



Конструкция if – else

Конструкция if – elif – else

7. Какие операторы сравнения используются в Python?

Больше (>)

Меньше (<)

Равно(==)

Не равно(!= или <>)

8. Что называется простым условием? Приведите примеры.

Логические выражения типа kByte >= 1023 являются простыми, так как в них выполняется только одна логическая операция.

```
>>> x = 8
>>> y = 13
>>> y < 15 and x
> 8 False
```

9. Что такое составное условие? Приведите примеры.

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

```
a = 3
if a == 3:
    print("hello 2")
```

10. Какие логические операторы допускаются при составлении сложных условий?

В таких случаях используются специальные операторы, объединяющие два и более простых логических выражения. Широко используются два оператора – так называемые логические И (and) и ИЛИ (or). Чтобы получить True при использовании оператора and, необходимо, чтобы результаты обоих простых выражений, которые связывает данный оператор, были истинными. Если хотя бы в одном случае результатом будет False, то и все сложное выражение будет ложным. Чтобы получить True при использовании оператора

or, необходимо, чтобы результат хотя бы одного простого выражения, входящего в состав сложного, был истинным. В случае оператора or сложное выражение становится ложным лишь тогда, когда ложны оба составляющие его простые выражения.

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

Не может

12. Какой алгоритм является алгоритмом циклической структуры?

Алгоритм циклической структуры — это алгоритм, в котором предусмотрено неоднократное выполнение одной и той же последовательности действий.

13. Типы циклов в языке Python.

Оператор цикла while

Операторы break и continue

14. Назовите назначение и способы применения функции range .

Функция range возвращает неизменяемую последовательность чисел в виде объекта range.

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

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

stop - до какого числа продолжается последовательность чисел. У

казанное число не включается в диапазон

step - с каким шагом растут числа. По умолчанию 1

16. Могут ли быть циклы вложенными?

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

18. Для чего нужен оператор break?

Оператор break предназначен для досрочного прерывания работы цикла while.

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

При работе с циклами используются операторы break и continue.

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

## 20. Для чего нужны стандартные потоки `stdout` и `stderr`?

Рассмотрим каким образом реализован вывод ошибок в приведенной выше программе. В операционной системе по умолчанию присутствуют стандартных потока вывода на консоль: буферизованный поток `stdout` для вывода данных и информационных сообщений, а также не буферизованный поток `stderr` для вывода сообщений об ошибках. По умолчанию функция `print` использует поток `stdout`.

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

Для того, чтобы использовать поток `stderr` необходимо передать его в параметре `file` функции `print`. Само же определение потоков `stdout` и `stderr` находится в стандартном пакете Python `sys`. 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.

## 22. Каково назначение функции `exit`?

Если в процессе выполнения программы произошли ошибки, программа должна передать операционной системе код возврата отличный от нуля. В Python завершить программу и передать операционной системе заданный код возврата можно посредством функции `exit`.