

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

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №4
дисциплины «Основы программной инженерии»

Выполнил:
Мелтонян Одиссей
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Проверил: Воронкин Р. А.

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г

Тема: Основы языка Python

Цель работы: исследование процесса установки и базовых возможностей языка Python.

Ход работы:

1. Изучил теоретический материал
2. Создал общедоступный репозиторий на GitHub, в котором была использована лицензия MIT и язык программирования Python.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*

Owner * odik8 / **Repository name *** BSE4

✔ BSE4 is available.

Great repository names are short and memorable. Need inspiration? How about [super-octo-enigma](#) ?

Description (optional)

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

☐ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: Python

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: MIT License

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

You are creating a public repository in your personal account.

[Create repository](#)

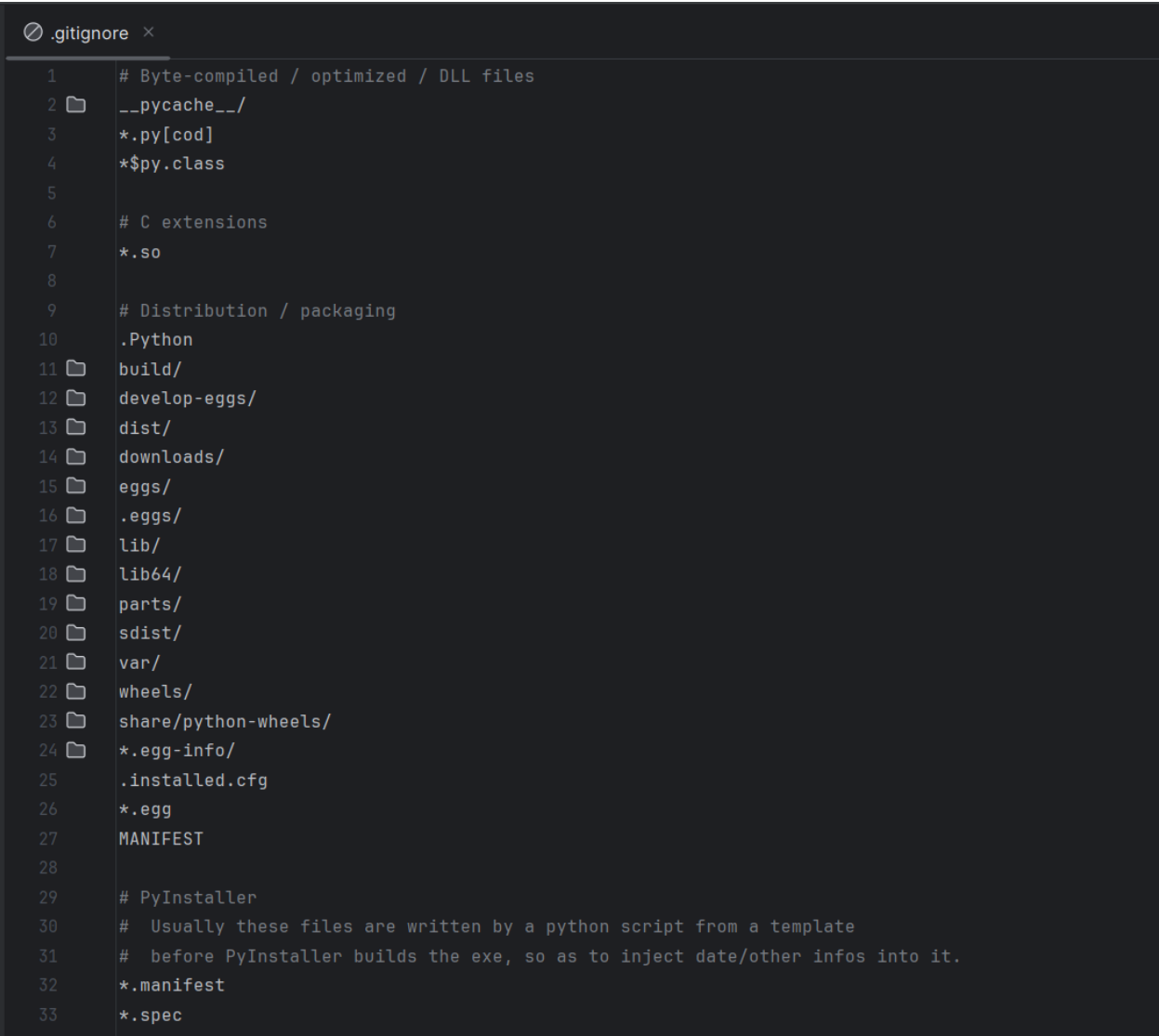
Рисунок 1. – Создание репозитория

3. Выполнил клонирование созданного репозитория.

```
varfe@DESKTOP-E108KEN MINGW64 ~/Рабочий стол
$ git clone https://github.com/odik8/BSE4.git
Cloning into 'BSE4'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
```

Рисунок 2. – Клонирование репозитория

4. Дополнил файл .gitignore необходимыми правилами для работы с IDE PyCharm.



```
.gitignore
1  # Byte-compiled / optimized / DLL files
2  __pycache__/
3  *.py[cod]
4  *$py.class
5
6  # C extensions
7  *.so
8
9  # Distribution / packaging
10 .Python
11 build/
12 develop-eggs/
13 dist/
14 downloads/
15 eggs/
16 .eggs/
17 lib/
18 lib64/
19 parts/
20 sdist/
21 var/
22 wheels/
23 share/python-wheels/
24 *.egg-info/
25 .installed.cfg
26 *.egg
27 MANIFEST
28
29 # PyInstaller
30 # Usually these files are written by a python script from a template
31 # before PyInstaller builds the exe, so as to inject date/other infos into it.
32 *.manifest
33 *.spec
```

Рисунок 3. – Файл .gitignore

5. Организовал свой репозиторий в соответствии с моделью ветвления git-flow. Для этого создал ветку develop из main. От ветки develop будут создаваться ветки features для каждой функции.

```
varfe@DESKTOP-E108KEN MINGW64 ~/Рабочий стол/bse4 (main)
$ git branch develop

varfe@DESKTOP-E108KEN MINGW64 ~/Рабочий стол/bse4 (main)
$ git checkout develop
Switched to branch 'develop'

varfe@DESKTOP-E108KEN MINGW64 ~/Рабочий стол/bse4 (develop)
$ |
```

Рисунок 4. – Создание ветки develop

6. Создал файл user.py

```
varfe@DESKTOP-E108KEN MINGW64 ~/Рабочий стол/bse4 (develop)
$ git checkout -b feature/user
Switched to a new branch 'feature/user'

varfe@DESKTOP-E108KEN MINGW64 ~/Рабочий стол/bse4 (feature/user)
$ touch user.py
```

Рисунок 5. – Создание файла user.py

7. Решил следующие задачи с помощью языка программирования Python3 и IDE PyCharm:

8. Написал программу (файл user.py), которая запрашивала у пользователя:

его имя (например, "What is your name?")

возраст ("How old are you?")

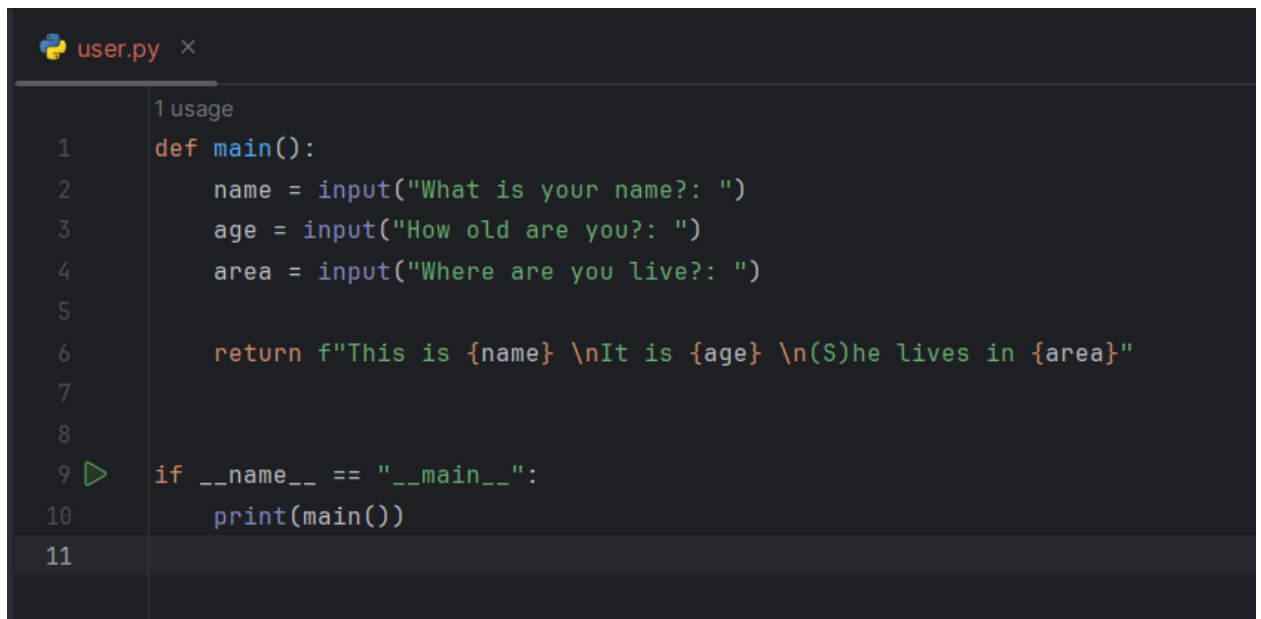
место жительства ("Where are you live?")

После этого выводила три строки:

"This is `имя`"

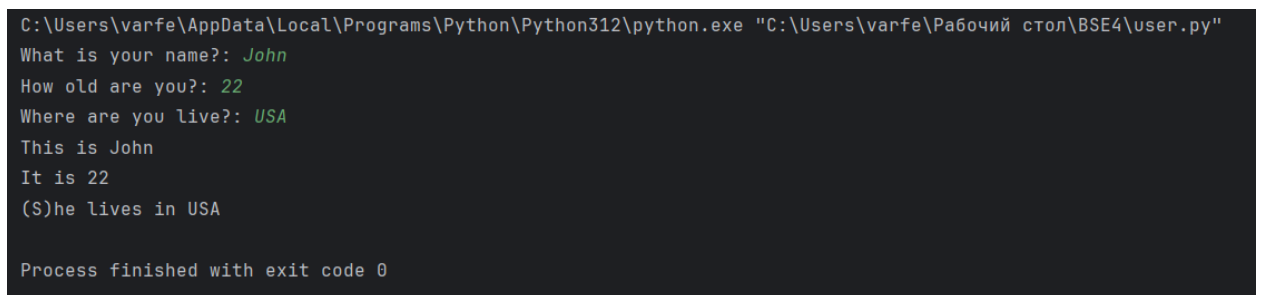
"It is `возраст`"

"(S)he lives in `место_жительства`"



```
1 usage
2 def main():
3     name = input("What is your name?: ")
4     age = input("How old are you?: ")
5     area = input("Where are you live?: ")
6
7     return f"This is {name} \nIt is {age} \n(S)he lives in {area}"
8
9 if __name__ == "__main__":
10     print(main())
11
```

Рисунок 6. – Код файла user.py

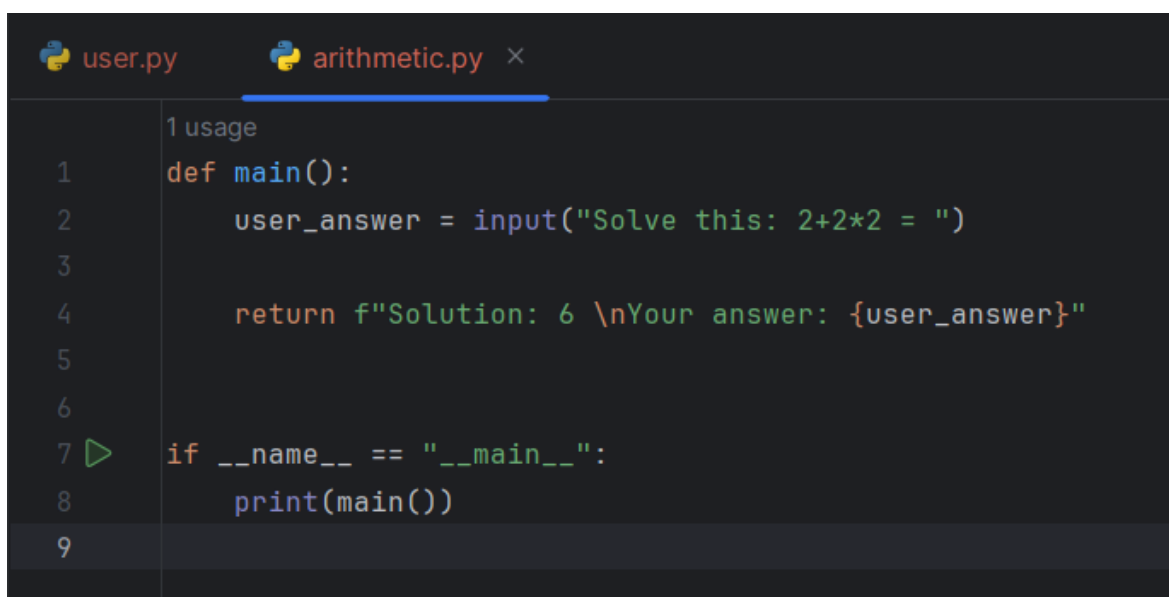


```
C:\Users\varfe\AppData\Local\Programs\Python\Python312\python.exe "C:\Users\varfe\Рабочий стол\BSE4\user.py"
What is your name?: John
How old are you?: 22
Where are you live?: USA
This is John
It is 22
(S)he lives in USA

Process finished with exit code 0
```

Рисунок 7. – Результат выполнения программы

9. Написал программу (файл arithmetic.py), которая предлагала бы пользователю решить пример $4 * 100 - 54$. Потом выводила бы на экран правильный ответ и ответ пользователя.



```
1 usage
2 def main():
3     user_answer = input("Solve this: 2+2*2 = ")
4
5     return f"Solution: 6 \nYour answer: {user_answer}"
6
7 if __name__ == "__main__":
8     print(main())
9
```

Рисунок 8. – Код файла arithmetic.py

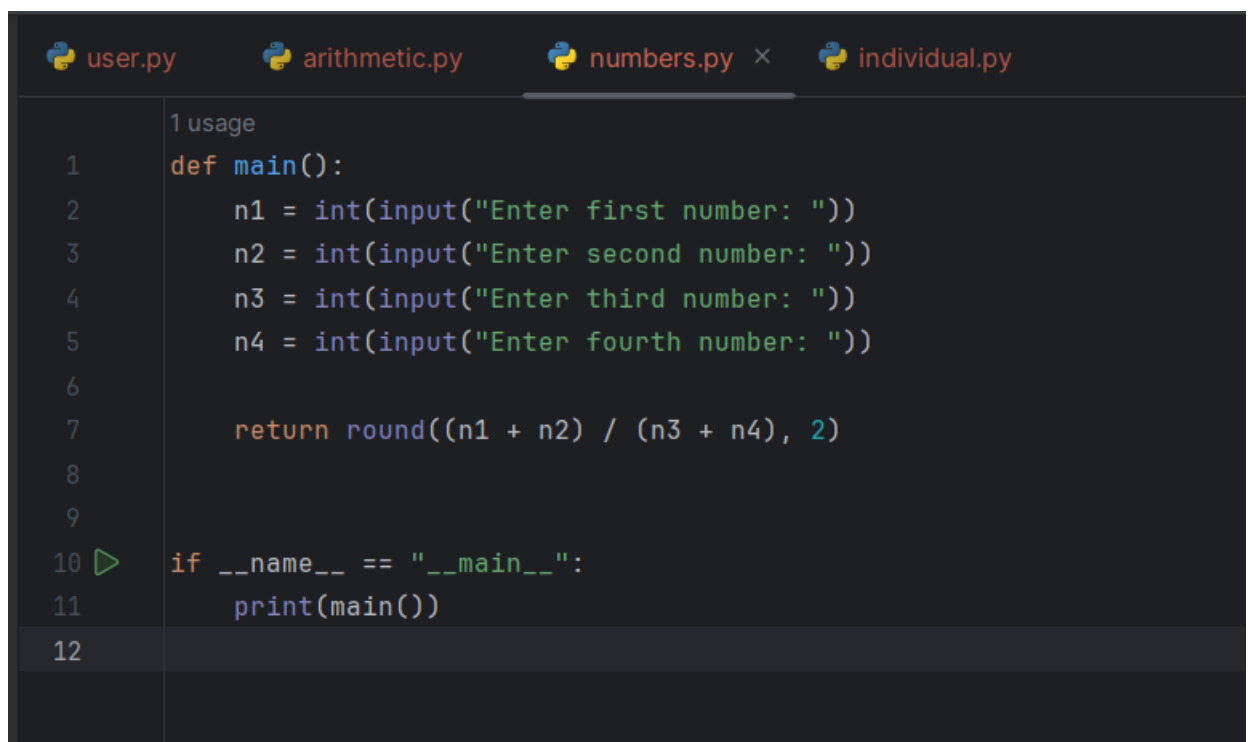
```
C:\Users\varfe\AppData\Local\Programs\Python\
Solve this: 2+2*2 = 5
Solution: 6
Your answer: 5

Process finished with exit code 0
```

Рисунок 9. – Результат выполнения программы

10. Запросите у пользователя четыре числа (файл numbers.py). Отдельно сложите первые два и отдельно вторые два. Разделите первую сумму на вторую. Выведите результат на экран так, чтобы ответ содержал две цифры после запятой.

Код:



```
user.py arithmetic.py numbers.py × individual.py
1 usage
2 def main():
3     n1 = int(input("Enter first number: "))
4     n2 = int(input("Enter second number: "))
5     n3 = int(input("Enter third number: "))
6     n4 = int(input("Enter fourth number: "))
7
8     return round((n1 + n2) / (n3 + n4), 2)
9
10 if __name__ == "__main__":
11     print(main())
12
```

Рисунок 10. – Файл numbers.py

```
C:\Users\varfe\AppData\Local\Programs\Python\Python312\
Enter first number: 1
Enter second number: 4
Enter third number: 6
Enter fourth number: 9
0.33

Process finished with exit code 0
|
```

Рисунок 11. – Результат выполнения программы

11. Написал программу (файл individual.py) для решения индивидуального задания. Вариант 14. Два автомобиля едут навстречу друг другу с постоянными скоростями V_1 и V_2 км/ч. Определить, через какое время автомобили встретятся, если расстояние между ними было S км.

Код:

```
user.py arithmetic.py numbers.py individual.py x
1 usage
2 def main():
3     distance = float(input("Enter the distance between cars in kilometers: "))
4     speed1 = float(input("Enter the speed of the first car in km/h: "))
5     speed2 = float(input("BEnter the speed of the second car in km/h: "))
6
7     return f"The cars will meet in, {distance / (speed1 + speed2)}, hours"
8
9 if __name__ == "__main__":
    print(main())
```

Рисунок 12. – Файл individual.py

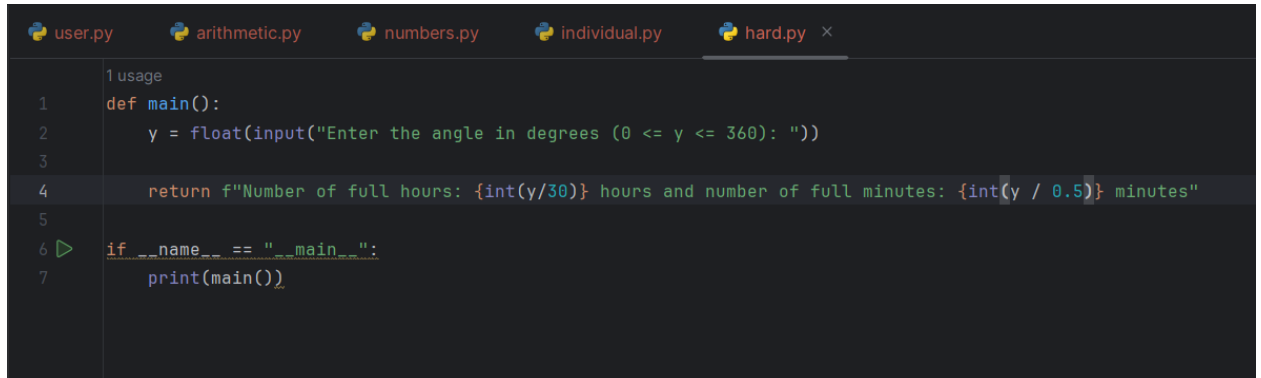
```
C:\Users\varfe\AppData\Local\Programs\Python\Python312\
Enter the distance between cars in kilometers: 100
Enter the speed of the first car in km/h: 30
BEnter the speed of the second car in km/h: 70
The cars will meet in, 1.0, hours

Process finished with exit code 0
```

Рисунок 13. – Результат выполнения программы

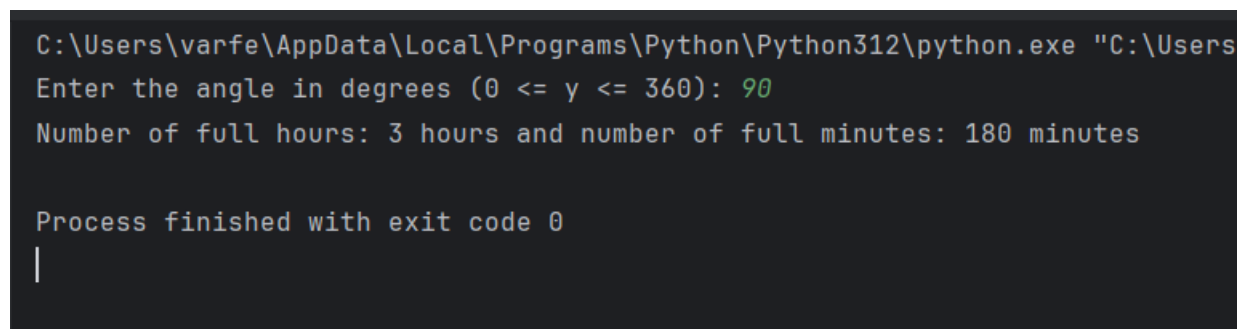
Задача повышенной сложности:

6. С начала суток часовая стрелка повернулась на градусов (, – вещественное число). Определить число полных часов и число полных минут, прошедших с начала суток.



```
user.py arithmetic.py numbers.py individual.py hard.py x
1 usage
2 def main():
3     y = float(input("Enter the angle in degrees (0 <= y <= 360): "))
4     return f"Number of full hours: {int(y/30)} hours and number of full minutes: {int(y / 0.5)} minutes"
5
6 if __name__ == "__main__":
7     print(main())
```

Рисунок 14. – Код решения

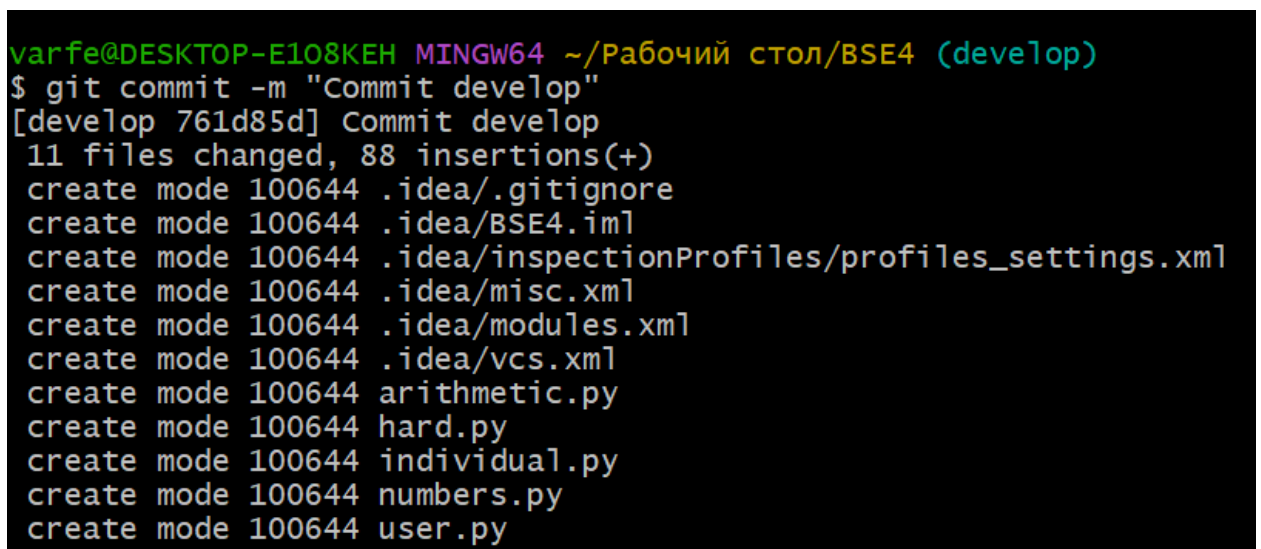


```
C:\Users\varfe\AppData\Local\Programs\Python\Python312\python.exe "C:\Users\
Enter the angle in degrees (0 <= y <= 360): 90
Number of full hours: 3 hours and number of full minutes: 180 minutes

Process finished with exit code 0
|
```

Рисунок 15. – Результат выполнения программы

12. Выполнил коммит файлов в репозиторий git в ветку для разработки.



```
varfe@DESKTOP-E108KEH MINGW64 ~/Рабочий стол/BSE4 (develop)
$ git commit -m "Commit develop"
[develop 761d85d] Commit develop
11 files changed, 88 insertions(+)
create mode 100644 .idea/.gitignore
create mode 100644 .idea/BSE4.iml
create mode 100644 .idea/inspectionProfiles/profiles_settings.xml
create mode 100644 .idea/misc.xml
create mode 100644 .idea/modules.xml
create mode 100644 .idea/vcs.xml
create mode 100644 arithmetic.py
create mode 100644 hard.py
create mode 100644 individual.py
create mode 100644 numbers.py
create mode 100644 user.py
```

Рисунок 16. – Коммит файлов

13. Выполнил слияние ветки для разработки с веткой master.


```

varfe@DESKTOP-E108KEH MINGW64 ~/Рабочий стол/BSE4 (main)
$ git checkout main
Already on 'main'
Your branch is up to date with 'origin/main'.

varfe@DESKTOP-E108KEH MINGW64 ~/Рабочий стол/BSE4 (main)
$ git merge develop
Updating c9b35f8..761d85d
Fast-forward
 .idea/.gitignore           | 8 ++++++++
 .idea/BSE4.iml             | 8 ++++++++
 .idea/inspectionProfiles/profiles_settings.xml | 6 ++++++
 .idea/misc.xml             | 7 ++++++++
 .idea/modules.xml          | 8 ++++++++
 .idea/vcs.xml              | 6 ++++++
 arithmetic.py              | 8 ++++++++
 hard.py                    | 7 ++++++++
 individual.py              | 9 ++++++++
 numbers.py                 | 11 ++++++++
 user.py                    | 10 ++++++++
11 files changed, 88 insertions(+)
create mode 100644 .idea/.gitignore
create mode 100644 .idea/BSE4.iml
create mode 100644 .idea/inspectionProfiles/profiles_settings.xml
create mode 100644 .idea/misc.xml
create mode 100644 .idea/modules.xml
create mode 100644 .idea/vcs.xml
create mode 100644 arithmetic.py
create mode 100644 hard.py
create mode 100644 individual.py
create mode 100644 numbers.py
create mode 100644 user.py

```

Рисунок 17. – Слияние веток

14. Отправил сделанные изменения на сервер GitHub.

```

varfe@DESKTOP-E108KEH MINGW64 ~/Рабочий стол/BSE4 (main)
$ git push origin main
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/odik8/BSE4.git
c9b35f8..761d85d  main -> main

```

Результат 18. – Финальный пуш

Вывод: были исследованы процессы установки и базовые возможности языка Python.

Вопросы для защиты работы:

1. Основные этапы установки Python в Windows и Linux:

- Windows:

1. Перейдите на официальный сайт Python (<https://www.python.org/>).
 2. Скачайте исполняемый установщик для Windows.
 3. Запустите установщик и следуйте инструкциям.
 4. Выберите опцию "Add Python to PATH" для удобства использования Python из командной строки.
 5. Завершите установку.
- Linux:
 1. Многие дистрибутивы Linux поставляются с предустановленным Python. Вы можете проверить его наличие с помощью команды **python --version**.
 2. Если Python не установлен, воспользуйтесь пакетным менеджером вашего дистрибутива, например, **apt** (для Debian/Ubuntu) или **yum** (для CentOS/RedHat), чтобы установить Python.
 3. Установите желаемую версию Python и необходимые пакеты.
2. Отличие Anaconda от официального Python: Anaconda - это дистрибуция Python, предназначенная для научных вычислений и анализа данных.
Основные отличия:
 - В Anaconda включены множество научных библиотек, таких как NumPy, SciPy, Pandas, и многие другие, что упрощает научные вычисления.
 - Anaconda имеет свой менеджер пакетов conda, который управляет зависимостями и позволяет создавать изолированные среды.
 - Официальный Python с сайта python.org поставляется с базовыми библиотеками, и вы должны устанавливать дополнительные пакеты вручную.
 3. Проверка работоспособности Anaconda:
 - Откройте командную строку или терминал.

- Введите **conda --version** для проверки версии conda.
- Введите **python --version** для проверки версии Python, установленной с Anaconda.

4. Задание интерпретатора в PyCharm:

- Откройте проект в PyCharm.
- Перейдите в "File" > "Settings" (или "PyCharm" > "Preferences" на macOS).
- В разделе "Project" выберите "Python Interpreter".
- Нажмите на значок шестеренки и выберите "Add".
- Выберите интерпретатор Python из Anaconda или другого источника и нажмите "ОК".

5. Запуск программы в PyCharm:

- Откройте файл с программой.
- Нажмите правой кнопкой мыши на коде и выберите "Run" или "Debug" в контекстном меню.
- Или используйте горячие клавиши (например, Shift + F10 для запуска).

6. Интерактивный и пакетный режимы работы Python:

- Интерактивный режим (REPL) позволяет выполнять команды по одной и сразу видеть результат.
- Пакетный режим используется для выполнения скриптов, которые сохраняются в файлах и выполняются последовательно.

7. Динамическая типизация в Python:

- В Python переменные могут изменять свой тип во время выполнения программы.

8. Основные типы данных в Python:

- int (целые числа)
- float (вещественные числа)
- str (строки)
- bool (логические значения)

- list (списки)
- tuple (кортежи)
- dict (словари)
- set (множества)

9. Создание объектов в памяти и операция присваивания:

- При объявлении переменной, Python создает объект в памяти, связывая его с именем переменной.
- Пример: `x = 5` создает объект с числом 5 и связывает его с именем "x".

10. Получение списка ключевых слов в Python:

- Используйте **import keyword** и **keyword.kwlist** для получения списка ключевых слов.

11. Функции **id()** и **type()**:

- **id(obj)** возвращает уникальный идентификатор объекта в памяти.
- **type(obj)** возвращает тип объекта.

12. Изменяемые и неизменяемые типы:

- Изменяемые типы могут быть изменены после создания (списки, словари).
- Неизменяемые типы не могут быть изменены после создания (целые числа, строки).

13. Различие между операциями деления и целочисленного деления:

- `/` выполняет деление и всегда возвращает float.
- `//` выполняет целочисленное деление и возвращает int.

14. Работа с комплексными числами:

- В Python комплексные числа представляются в форме **a + bj**.
- Используйте **complex(real, imag)** для создания комплексного числа.
- Модуль **cmath** предоставляет функции для работы с комплексными числами.

15. Библиотека **math** и **cmath**:

- **math** предоставляет математические функции для работы с вещественными числами.
- **cmath** предоставляет аналогичные функции для комплексных чисел.

16. Параметры **sep** и **end** в **print()**:

- **sep** определяет разделитель между аргументами функции **print()**.
- **end** определяет символ, добавляемый в конец вывода.

17. Метод **format()** и форматирование строк:

- **format()** используется для вставки значений в строки.
- Другие средства форматирования включают использование **%**, **f**-строки и метод **.format()**.

18. Ввод значений с консоли:

- Используйте **input()** для ввода строк с клавиатуры.
- Преобразуйте введенные строки в нужный тип данных (**int** или **float**) при необходимости.