

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 №10
дисциплины «Анализ данных»

Выполнил:
Гайчук Дарья Дмитриевна
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика
и вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

(подпись)

Руководитель практики:
Воронкин Р.А.-доцент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

Цель работы: приобретение навыков написания многопоточных приложений на языке программирования Python версии 3.x.

Порядок выполнения работы:

1. Создала новый репозиторий и клонировала его на свой компьютер.

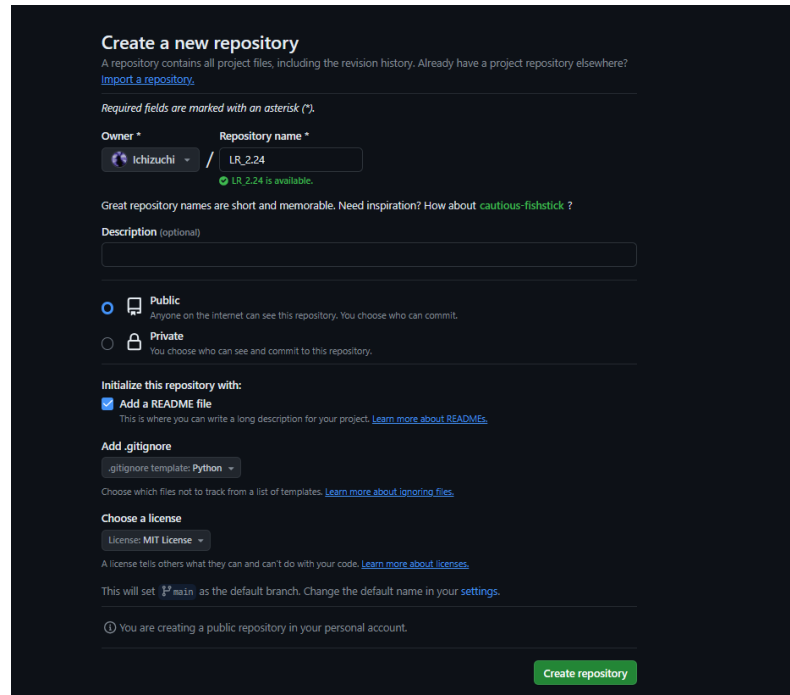


Рисунок 1. Создан новый репозиторий

2. Клонировала репозиторий на свой компьютер.

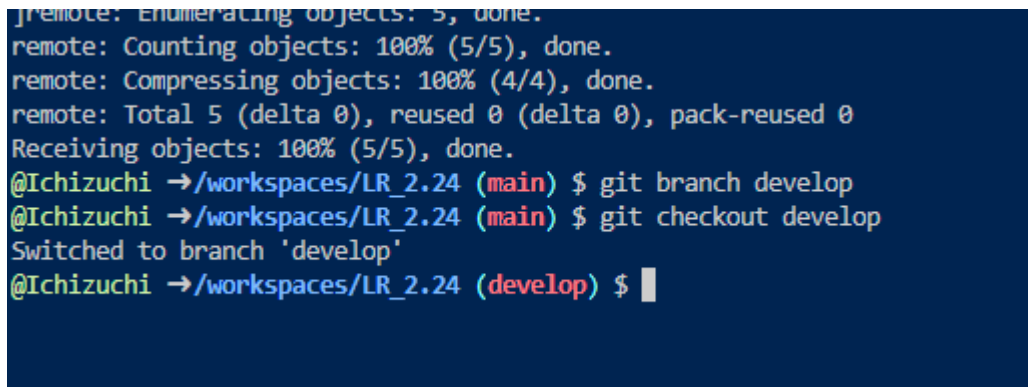


Рисунок 2. Клонирование и модель ветвления git-flow

3. Создала виртуальное окружение Anaconda с именем репозитория.

```

● @Ichizuchi →/workspaces/LR_2.24 (develop) $ conda create -n myenv python=3.10
Retrieving notices: ...working... done
Channels:
 - defaults
Platform: linux-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

environment location: /opt/conda/envs/myenv

added / updated specs:
 - python=3.10

The following packages will be downloaded:

package | build
-----|-----
_libgcc_mutex-0.1 | main 3 KB
_openmp_mutex-5.1 | 1_gnu 21 KB
bzip2-1.0.8 | h5eee18b_5 262 KB
ca-certificates-2024.3.11 | h06a4308_0 127 KB
ld_impl_linux-64-2.38 | h1181459_1 654 KB
libffi-3.4.4 | h6a678d5_0 142 KB
libgcc-ng-11.2.0 | h1234567_1 5.3 MB

```

Рисунок 3. Создание виртуального окружения

Работа с примерами

```

Код > primer1.py > ...
4 # В этом примере мы создаем функцию order_processor, которая может реализовывать в себе бизнес логику, например,
5 # обработку заказа. При этом, если она получает сообщение stop, то прекращает свое выполнение. В главном потоке мы
6 # создаем и запускаем три потока для обработки заказов. Запущенные потоки видят, что очередь пуста и "встают на
7 # блокировку" при вызове wait(). В главном потоке в очередь добавляются десять заказов и сообщения для остановки
8 # обработчиков, после этого вызывается метод notify_all() для оповещения всех заблокированных потоков о том, что
9 # данные для обработки есть в очереди.
10
11 from threading import Condition, Thread
12 from queue import Queue
13 from time import sleep
14
15 cv = Condition()
16 q = Queue()
17 # Consumer function for order processing
18 def order_processor(name):
19     while True:
20         with cv:
21             # Wait while queue is empty
22             while q.empty():
23                 cv.wait()
24             try:
25                 # Get data (order) from queue
26                 order = q.get_nowait()
27                 print(f"{name}: {order}")
28                 # If get "stop" message then stop thread
29                 if order == "stop":
30                     break
31             except:
32                 pass

```

ПРОБЛЕМЫ Выходные данные КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ ПОРТЫ 1 КОММЕНТАРИИ

```

/home/codespace/.python/current/bin/python3 /workspaces/LR_2.24/Kon/primer1.py
● @Ichizuchi →/workspaces/LR_2.24 (develop) $ /home/codespace/.python/current/bin/python3 /workspaces/LR_2.24/Kon/primer1.py
thread 1: order 0
thread 1: order 1
thread 1: order 2
thread 1: order 3
thread 1: order 4
thread 1: order 5
thread 1: order 6
thread 1: order 7
thread 1: order 8
thread 1: order 9
thread 1: stop
thread 3: stop
thread 2: stop
● @Ichizuchi →/workspaces/LR_2.24 (develop) $

```

Рисунок 4. Результат выполнения примера №1

```
Код > primer2.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  # Пример работы с Event-объектом
5
6  from threading import Thread, Event
7  from time import sleep, time
8
9  event = Event()
10
11
12  def worker(name: str):
13      event.wait()
14      print(f"Worker: {name}")
15
16
17  if __name__ == "__main__":
18      # Clear event
19      event.clear()
20      # Create and start workers
21      workers = [Thread(target=worker, args=(f"wrk {i}",)) for i in range(5)]
22      for w in workers:
23          w.start()
24      print("Main thread")
25      event.set()
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ ПОРТЫ 1 КОММЕНТАРИИ

```
thread 1: order 3
thread 1: order 4
thread 1: order 5
thread 1: order 6
thread 1: order 7
thread 1: order 8
thread 1: order 9
thread 1: stop
thread 3: stop
thread 2: stop
@Ichizuchi →/workspaces/LR_2.24 (develop) $ /home/codespace/.python/current/bin/python3 /workspaces/LR_2.24/Код/primer2.py
Main thread
Worker: wrk 0
Worker: wrk 2
Worker: wrk 3
Worker: wrk 1
Worker: wrk 4
@Ichizuchi →/workspaces/LR_2.24 (develop) $
```

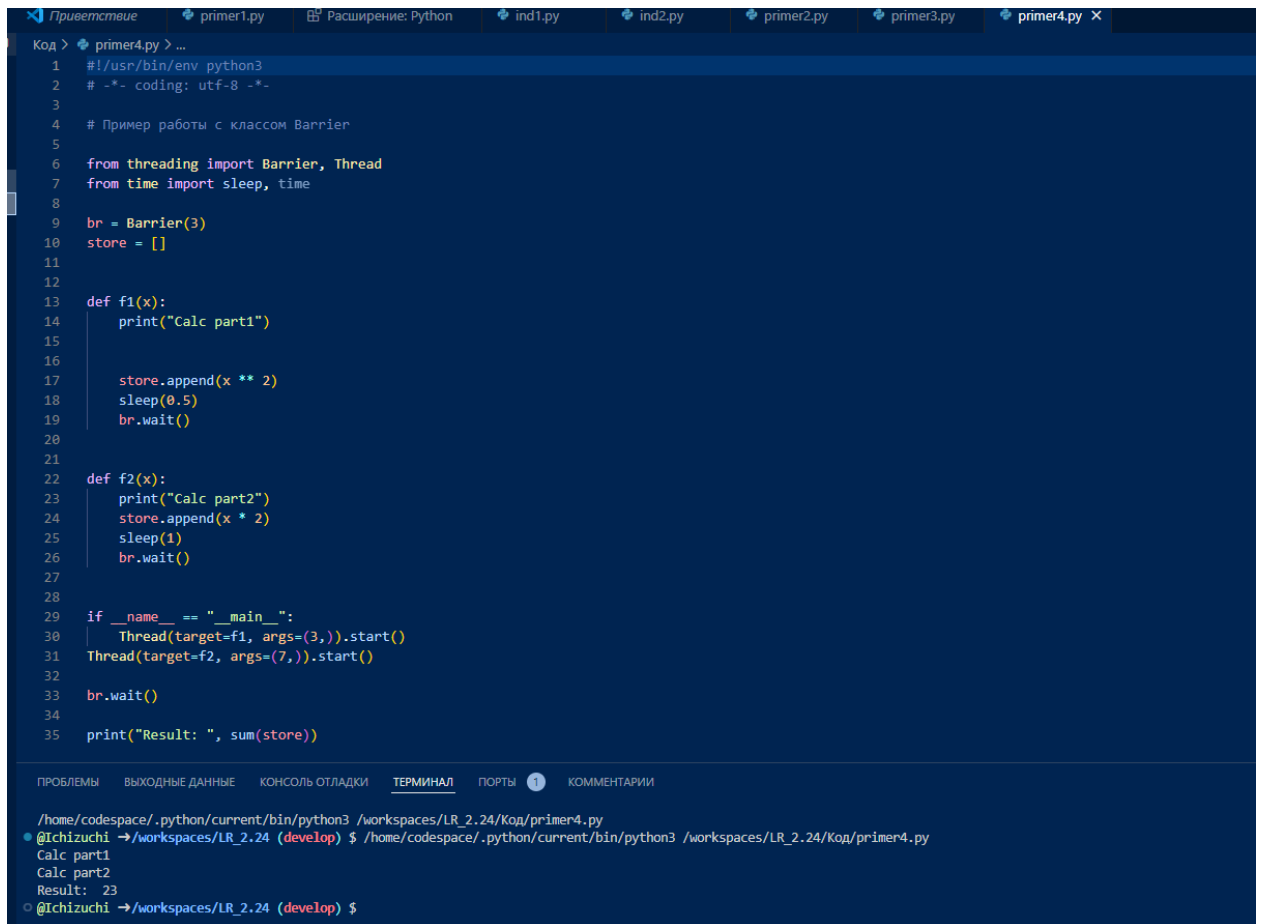
Рисунок 5. Результат выполнения примера №2

```
Код > primer3.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  #Пример работы с таймером
5
6  if __name__ == "__main__":
7      from threading import Timer
8      from time import sleep, time
9      timer = Timer(interval=3, function=lambda: print("Message from Timer!"))
10     timer.start()
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ ПОРТЫ 1 КОММЕНТАРИИ

```
/home/codespace/.python/current/bin/python3 /workspaces/LR_2.24/Код/primer3.py
@Ichizuchi →/workspaces/LR_2.24 (develop) $ /home/codespace/.python/current/bin/python3 /workspaces/LR_2.24/Код/primer3.py
Message from Timer!
@Ichizuchi →/workspaces/LR_2.24 (develop) $
```

Рисунок 6. Результат выполнения примера №3



```
Код > primer4.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  # Пример работы с классом Barrier
5
6  from threading import Barrier, Thread
7  from time import sleep, time
8
9  br = Barrier(3)
10 store = []
11
12
13 def f1(x):
14     print("Calc part1")
15
16     store.append(x ** 2)
17     sleep(0.5)
18     br.wait()
19
20
21
22 def f2(x):
23     print("Calc part2")
24     store.append(x * 2)
25     sleep(1)
26     br.wait()
27
28
29 if __name__ == "__main__":
30     Thread(target=f1, args=(3,)).start()
31     Thread(target=f2, args=(7,)).start()
32
33     br.wait()
34
35     print("Result: ", sum(store))
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ **ТЕРМИНАЛ** ПОРТЫ КОММЕНТАРИИ

```
/home/codespace/.python/current/bin/python3 /workspaces/LR_2.24/Код/primer4.py
@Ichizuchi → /workspaces/LR_2.24 (develop) $ /home/codespace/.python/current/bin/python3 /workspaces/LR_2.24/Код/primer4.py
Calc part1
Calc part2
Result: 23
@Ichizuchi → /workspaces/LR_2.24 (develop) $
```

Рисунок 7. Результат выполнения примера №4

Выполнение индивидуального задания

Разработать приложение, в котором выполнить решение вычислительной задачи (например, задачи из области физики, экономики, математики, статистики и т. д.) с помощью паттерна “Производитель-Потребитель”, условие которой предварительно необходимо согласовать с преподавателем.

```
7  Разработать приложение, в котором выполнить решение вычислительной задачи
8  (например, задачи из области физики, экономики, математики, статистики и т. д.) с помощью
9  паттерна "Производитель-Потребитель", условие которой предварительно необходимо
10 согласовать с преподавателем
11 """
12 """
13 Взята задача обработок посылок на почте.
14 """
15 queue = Queue()
16
17
18 class Package():
19     def __init__(self, weight, tip):
20         self.id = random.randint(1, 1000)
21         self.weight = weight
22         self.tip = tip
23
24     def handling(self):
25         if self.weight >= 1000:
26             print("Большой пакет, цена отправки - ", round(self.weight * 0.7, 2))
27         else:
28             print("Мелкий пакет, цена отправки - ", round(self.weight * 0.8, 2))
29         print("-----")
30
31
32 class ProducerThread(Thread):
33     def run(self):
34         counter = 0
35         while counter < 20:
36             queue.put(parsels[counter])
37             print(f"Получена посылка, id - {parsels[counter].id} \n")
38             time.sleep(0.1)
39             counter += 1
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ ПОРТЫ 1 КОММЕНТАРИИ

```
Большой пакет, цена отправки - 1392.3
-----
Получена посылка, id - 252

Посылка простая
Мелкий пакет, цена отправки - 572.0
-----
Получена посылка, id - 62

Посылка с наложным платежом
Мелкий пакет, цена отправки - 772.8
-----
@Ichizuchi →/workspaces/LR_2.24 (develop) $
```

Рисунок 8. Результат выполнения

Для своего индивидуального задания лабораторной работы 2.23 необходимо организовать конвейер, в котором сначала в отдельном потоке вычисляется значение первой функции, после чего результаты вычисления должны передаваться второй функции, вычисляемой в отдельном потоке. Потоки для вычисления значений двух функций должны запускаться одновременно.

```
Код > ind2.py > ...
6   Для своего индивидуального задания лабораторной работы 2.23 необходимо
7   организовать конфейер, в котором сначала в отдельном потоке вычисляется значение
8   первой функции, после чего результаты вычисления должны передаваться второй функции,
9   вычисляемой в отдельном потоке. Потоки для вычисления значений двух функций должны
10  запускаться одновременно
11  """
12
13  CONST_PRECISION = 1e-07
14  qe = Queue()
15  lock = Lock()
16
17
18  def sum(x=-0.7):
19      lock.acquire()
20      n, s, m, curr = 0, 0, 0, 0
21      while True:
22          pre = (n + 1) * x**n
23          n += 1
24          curr = (n + 1) * x**n
25          if abs(curr - pre) < CONST_PRECISION:
26              break
27          s += curr
28          qe.put(s)
29      lock.release()
30
31
32  def func_y(x):
33      result = 1/(math.pow((1 - x), 2))
34      print(result)
35
36
37  if __name__ == '__main__':
38      t1 = Thread(target=sum).start()
39      t2 = Thread(target=func_y(qe.get())).start()
40
41
42  ПРОБЛЕМЫ    ВЫХОДНЫЕ ДАННЫЕ    КОНСОЛЬ ОТЛАДКИ    ТЕРМИНАЛ    ПОРТЫ    КОММЕНТАРИИ
/home/codespace/.python/current/bin/python3 /workspaces/LR_2.24/Код/ind2.py
@Ichizuchi → /workspaces/LR_2.24 (develop) $ /home/codespace/.python/current/bin/python3 /workspaces/LR_2.24/Код/ind2.py
0.17361111111111111
@Ichizuchi → /workspaces/LR_2.24 (develop) $
```

Рисунок 9. Результат выполнения

4. Сформировала файлы environment.yml и requirements.txt

```
@Ichizuchi → /workspaces/LR_2.24 (develop) $ conda env export > environment.yml
@Ichizuchi → /workspaces/LR_2.24 (develop) $ pip freeze > requirements.txt
@Ichizuchi → /workspaces/LR_2.24 (develop) $ conda init
no change      /opt/conda/condabin/conda
no change      /opt/conda/bin/conda
no change      /opt/conda/bin/conda-env
no change      /opt/conda/bin/activate
no change      /opt/conda/bin/deactivate
no change      /opt/conda/etc/profile.d/conda.sh
no change      /opt/conda/etc/fish/conf.d/conda.fish
no change      /opt/conda/shell/condabin/Conda.psm1
no change      /opt/conda/shell/condabin/conda-hook.ps1
no change      /opt/conda/lib/python3.12/site-packages/conda-hook.ps1
```

Рисунок 10. Файлы environment.yml и requirements.txt

Ответы на контрольные вопросы

1. Lock-объект может находиться в двух состояниях: захваченное (заблокированное) и не захваченное (не заблокированное, свободное). После создания он находится в свободном состоянии. Для работы с Lock-объектом

используются методы `acquire()` и `release()`. Если `Lock` свободен, то вызов метода `acquire()` переводит его в заблокированное состояние. Повторный вызов `acquire()` приведет к блокировке инициировавшего это действие потока до тех пор, пока `Lock` не будет разблокирован каким-то другим потоком с помощью метода `release()`. Вывоз метода `release()` на свободном `Lock`-объекте приведет к выбросу исключения `RuntimeError`.

2. В отличие от рассмотренного выше `Lock`-объекта `RLock` может освободить только тот поток, который его захватил. Повторный захват потоком уже захваченного `RLock`-объекта не блокирует его. `RLock`-объекты поддерживают возможность вложенного захвата, при этом освобождение происходит только после того, как был выполнен `release()` для внешнего `acquire()`. Сигнатуры и назначение методов `release()` и `acquire()` `RLock`-объектов совпадают с приведенными для `Lock`, но в отличие от него у `RLock` нет метода `locked()`. `RLock`-объекты поддерживают протокол менеджера контекста. С помощью команды закрытия `close()`.

3. Порядок работы с условными переменными выглядит так:

- На стороне `Consumer`'а: проверить доступен ли ресурс, если нет, то перейти в режим ожидания с помощью метода `wait()`, и ожидать оповещение от `Producer`'а о том, что ресурс готов и с ним можно работать. Метод `wait()` может быть вызван с таймаутом, по истечении которого поток выйдет из состояния блокировки и продолжит работу.
- На стороне `Producer`'а: произвести работы по подготовке ресурса, после того, как ресурс готов оповестить об этом ожидающие потоки с помощью методов `notify()` или `notify_all()`. Разница между ними в том, что `notify()` разблокирует только один поток (если он вызван без параметров), а `notify_all()` все потоки, которые находятся в режиме ожидания. Чтобы обновить данные в таблице, просто создайте соединение, затем создайте объект курсора с помощью соединения и, наконец, используйте оператор `UPDATE`.

4. При создании объекта Condition вы можете передать в конструктор объект Lock или RLock, с которым хотите работать. Перечислим методы объекта Condition с кратким описанием: `acquire(*args)` – захват объекта-блокировки. `release()` – освобождение объекта-блокировки. `wait(timeout=None)` – блокировка выполнения потока до оповещения о снятии блокировки. Через параметр `timeout` можно задать время ожидания оповещения о снятии блокировки. Если вызвать `wait()` на Условной переменной, у которой предварительно не был вызван `acquire()`, то будет выброшено исключение `RuntimeError`.

5. Чтобы перечислить все таблицы в базе данных SQLite3, вы должны запросить данные из таблицы `sqlite_master`, а затем использовать `fetchall()` для получения результатов из инструкции `SELECT`

6. При создании таблицы мы должны убедиться, что она еще не существует. Аналогично, при удалении/удалении таблицы она должна существовать. Чтобы проверить, не существует ли таблица уже, мы используем `IF NOT EXISTS` с оператором `CREATE TABLE` следующим образом.

7. Метод `executemany` можно использовать для вставки нескольких строк одновременно.

8. В базе данных Python SQLite3 мы можем легко хранить дату или время, импортируя модуль `datetime`. Следующие форматы являются наиболее часто используемыми форматами для `datetime`:

Вывод: в результате выполнения работы были приобретены навыки использования примитивов синхронизации в языке программирования Python версии 3.x.