

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ**

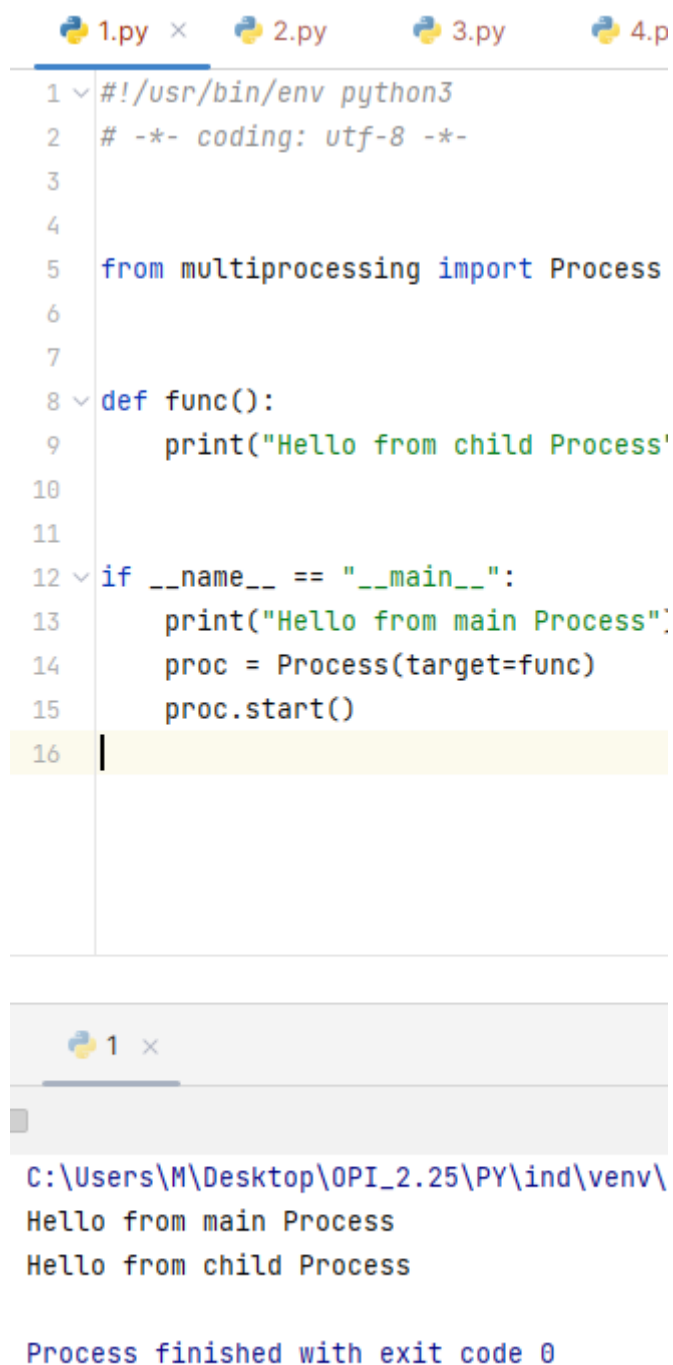
**Отчет по лабораторной работе № 2.25
«Управление процессами в Python»**

по дисциплине «Основы программной инженерии»

Выполнила:
Образцова Мария Дмитриевна,
2 курс, группа ПИЖ-б-о-21-1,
Проверил:
Доцент кафедры инфокоммуникаций,
Воронкин Р.А.

Ставрополь, 2023 г.

Методика и порядок выполнения работы



```
1.py x 2.py 3.py 4.p
1 1 ✓ #!/usr/bin/env python3
2 2 # -*- coding: utf-8 -*-
3 3
4 4
5 5 from multiprocessing import Process
6 6
7 7
8 8 ✓ def func():
9 9     print("Hello from child Process")
10 10
11 11
12 12 ✓ if __name__ == "__main__":
13 13     print("Hello from main Process")
14 14     proc = Process(target=func)
15 15     proc.start()
16 16
```

1 x

```
C:\Users\M\Desktop\OPI_2.25\PY\ind\venv\
Hello from main Process
Hello from child Process

Process finished with exit code 0
```

Рисунок 1 – Создание и ожидание завершения работы процессов

```
2.py x 3.py 4.py 5.py
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  from multiprocessing import Process
6
7
8  def func():
9      print("Hello from child Process")
10
11
12  if __name__ == "__main__":
13      print("Hello from main Process")
14      proc = Process(target=func)
15      proc.start()
16      print(f"Proc is_alive status: {proc.is_alive()}")
17      proc.join()
18      print("Goodbye")
19      print(f"Proc is_alive status: {proc.is_alive()}")
20
21  if __name__ == "__main__":
```

un 2 x

```
C:\Users\M\Desktop\OPI_2.25\PY\ind\venv\Scripts\python
Hello from main Process
Proc is_alive status: True
Hello from child Process
Goodbye
Proc is_alive status: False

Process finished with exit code 0
```

Рисунок 2 – Создание и ожидание завершения работы процессов

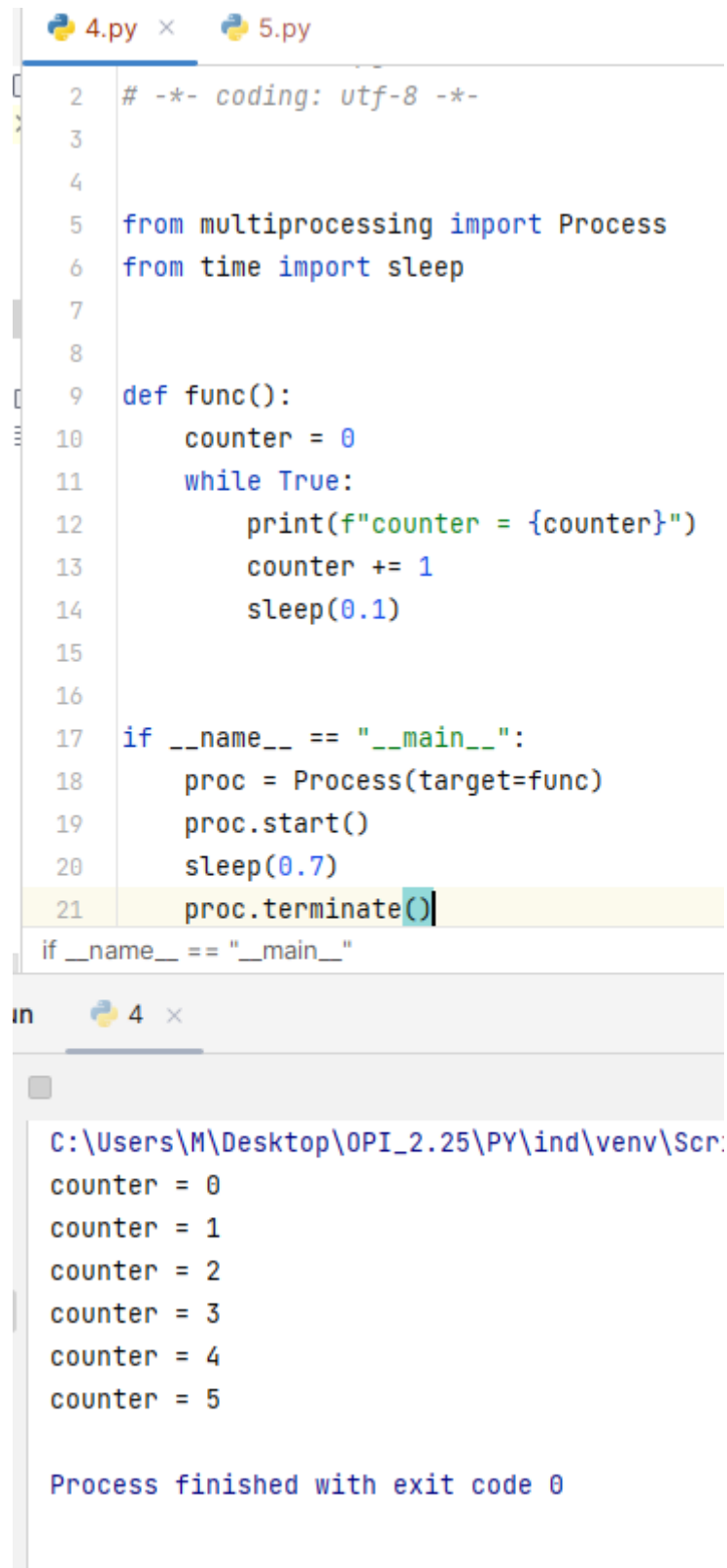
The image shows a Python IDE with three tabs: 3.py, 4.py, and 5.py. The active tab is 3.py, which contains the following code:

```
3
4
5 from multiprocessing import Process
6 from time import sleep
7
8
9 class CustomProcess(Process):
10     def __init__(self, limit):
11         Process.__init__(self)
12         self._limit = limit
13
14     def run(self):
15         for i in range(self._limit):
16             print(f"From CustomProcess: {i}")
17             sleep(0.5)
18
19
20 if __name__ == "__main__":
21     cpr = CustomProcess(3)
22     cpr.start()
23
24 if __name__ == "__main__":
```

The output window shows the following text:

```
run 3 x
C:\Users\M\Desktop\OPI_2.25\PY\ind\venv\Scripts\pytho
From CustomProcess: 0
From CustomProcess: 1
From CustomProcess: 2
Process finished with exit code 0
```

Рисунок 3 – Создание классов-наследников от Process



```
4.py x 5.py
2 # -*- coding: utf-8 -*-
3
4
5 from multiprocessing import Process
6 from time import sleep
7
8
9 def func():
10     counter = 0
11     while True:
12         print(f"counter = {counter}")
13         counter += 1
14         sleep(0.1)
15
16
17 if __name__ == "__main__":
18     proc = Process(target=func)
19     proc.start()
20     sleep(0.7)
21     proc.terminate()
22
23 if __name__ == "__main__":
```

in 4 x

```
C:\Users\M\Desktop\OPI_2.25\PY\ind\venv\Scr:
counter = 0
counter = 1
counter = 2
counter = 3
counter = 4
counter = 5

Process finished with exit code 0
```

Рисунок 4 – Принудительное завершение работы процессов

```
5.py x
4
5 from multiprocessing import Process
6 from time import sleep
7
8
9
10 def func(name):
11     counter = 0
12     while True:
13         print(f"proc {name}, counter = {counter}")
14         counter += 1
15         sleep(0.1)
16
17 if __name__ == "__main__":
18     proc1 = Process(target=func, args=("proc1",), da
19     proc2 = Process(target=func, args=("proc2",))
20     proc2.daemon = True
21     proc1.start()
22     proc2.start()
23     sleep(0.3)
if __name__ == "__main__"
```

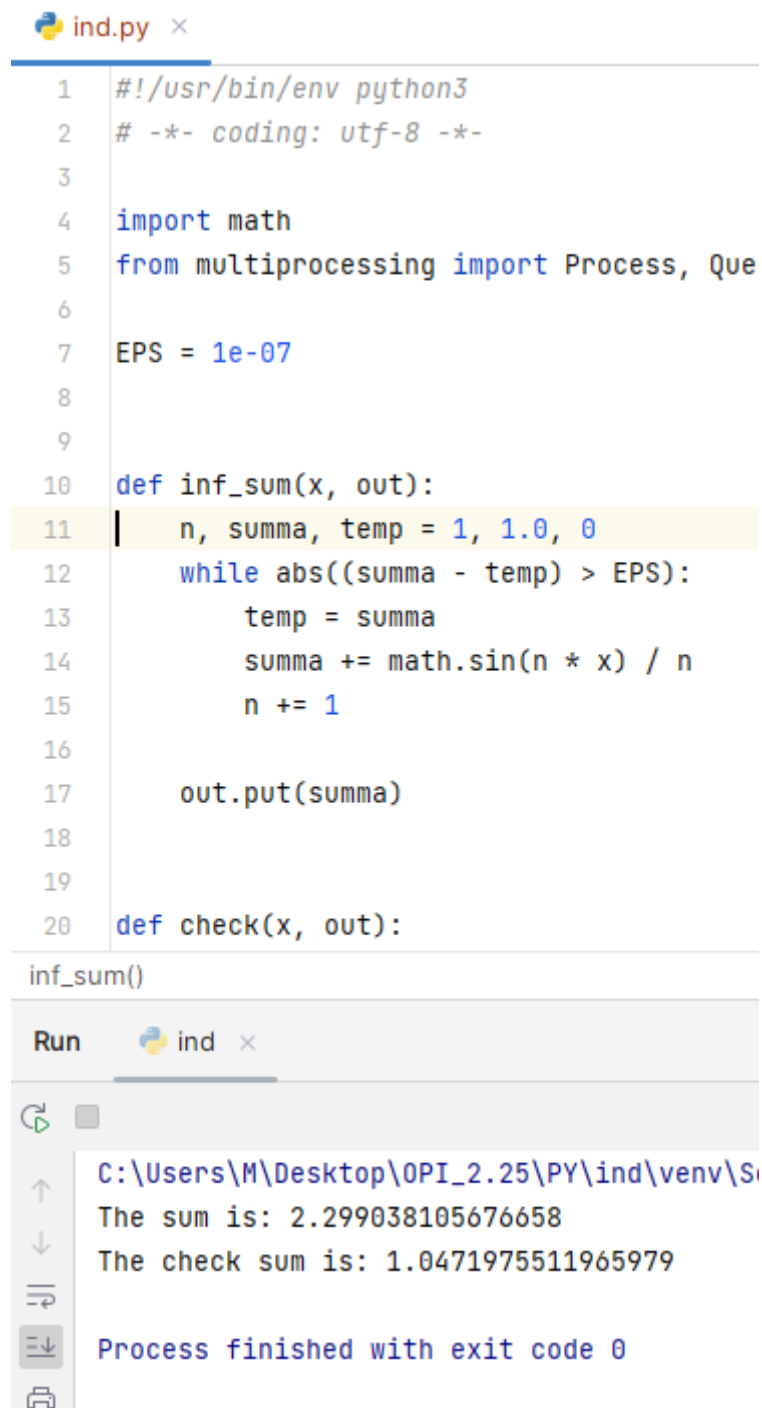
Run 5 x

```
C:\Users\M\Desktop\OPI_2.25\PY\ind\venv\Scripts\python.exe
proc proc1, counter = 0
proc proc2, counter = 0
proc proc1, counter = 1
proc proc2, counter = 1

Process finished with exit code 0
```

Рисунок 5 – Процессы-демоны

Для своего индивидуального задания лабораторной работы 2.23 необходимо реализовать вычисление значений в двух функций в отдельных процессах.



```
ind.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import math
5  from multiprocessing import Process, Queue
6
7  EPS = 1e-07
8
9
10 def inf_sum(x, out):
11     n, summa, temp = 1, 1.0, 0
12     while abs((summa - temp) > EPS):
13         temp = summa
14         summa += math.sin(n * x) / n
15         n += 1
16
17     out.put(summa)
18
19
20 def check(x, out):
    inf_sum()
```

Run ind x

C:\Users\M\Desktop\OPI_2.25\PY\ind\venv\Scripts>python ind.py
The sum is: 2.299038105676658
The check sum is: 1.0471975511965979
Process finished with exit code 0

Рисунок 6 – Индивидуальное задание

ВОПРОСЫ

1. Как создаются и завершаются процессы в Python?

`proc = Process(target=func)`

`proc.start()`

`join()` для того, чтобы программа ожидала завершения процесса.

Процессы завершаются при завершении функции, указанной в `target`, либо принудительно с помощью `kill()`, `terminate()`

2. В чем особенность создания классов-наследников от `Process`?

В классе наследнике от `Process` необходимо переопределить метод `run()` для того, чтобы он (класс) соответствовал протоколу работы с процессами.

3. Как выполнить принудительное завершение процесса?

В отличие от потоков, работу процессов можно принудительно завершить, для этого класс `Process` предоставляет набор методов:

`terminate()` - принудительно завершает работу процесса. В Unix отправляется команда `SIGTERM`, в Windows используется функция `TerminateProcess()`.

`kill()` - метод аналогичный `terminate()` по функционалу, только вместо `SIGTERM` в Unix будет отправлена команда `SIGKILL`.

4. Что такое процессы-демоны? Как запустить процесс-демон?

Процессы демоны по своим свойствам похожи на потоки-демоны, их суть заключается в том, что они завершают свою работу, если завершился родительский процесс.

Указание на то, что процесс является демоном должно быть сделано до его запуска (до вызова метода `start()`). Для демонического процесса запрещено самостоятельно создавать дочерние процессы. Эти процессы не являются демонами (сервисами) в понимании Unix, единственное их свойство – это завершение работы вместе с родительским процессом.

```
proc1 = Process(target=func, args=("proc1",), daemon=True)
```

```
proc2.daemon = True
```

```
proc1.start()
```

```
proc2.start()
```