

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

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №8
дисциплины
«Объектно-ориентированное программирование»
Вариант № 3

Выполнил:
Левашев Тимур Рашидович
3 курс, группа ИВТ-б-о-23-2,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

(подпись)

Проверил:
Доцент департамента цифровых,
робототехнических систем и
электроники института перспективной
инженерии Воронкин Р.А

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2025 г.

Тема работы: “Управление процессами в Python”

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

Порядок выполнения работы:

Ссылка на Git репозиторий: https://github.com/mazy99/oop_pract_9

1. Пример 1: Создание и ожидание завершения работы процесса.

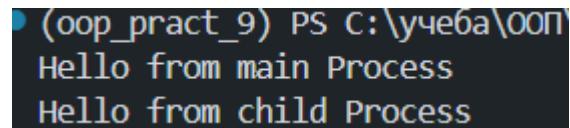
Листинг программы:

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

from multiprocessing import Process

def func():
    print("Hello from child Process")

if __name__ == "__main__":
    print("Hello from main Process")
    proc = Process(target=func)
    proc.start()
```



```
(oop_pract_9) PS C:\учеба\ооп
Hello from main Process
Hello from child Process
```

Рисунок 1 – Результат выполнения программы

2. Пример 2: Создание и ожидание завершения дочернего процесса с использованием метода `join()`, который приостанавливает выполнение главного процесса до завершения соответствующего дочернего процесса.

Листинг программы:

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

from multiprocessing import Process
```

```

def func():
    print("Hello from child Process")

if __name__ == "__main__":
    print("Hello from main Process")
    proc = Process(target=func)
    proc.start()
    proc.join()
    print("Goodbye")

```

```

● (oop_pract_9) PS C:\учеба\ооп\9 лаба\
Hello from main Process
Hello from child Process
Goodbye
● (oop_pract_9) PS C:\учеба\ооп\9 лаба\

```

Рисунок 2 – Результат выполнения программы

3. Пример 3: Проверка состояния дочернего процесса и ожидание его завершения с помощью метода `is_alive()`.

Листинг программы:

```

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

from multiprocessing import Process

def func():
    print("Hello from child Process")

if __name__ == "__main__":
    print("Hello from main Process")
    proc = Process(target=func)

    proc.start()

    print(f"Proc is_alive status: {proc.is_alive()}")

    proc.join()

    print("Goodbye")
    print(f"Proc is_alive status: {proc.is_alive()}")

```

```
Hello from main Process
Proc is_alive status: True
Hello from child Process
Goodbye
Proc is_alive status: False
```

Рисунок 3 – Результат выполнения программы

4. Пример 4: Создание классов-наследников от Process.

Листинг программы:

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

from multiprocessing import Process

class CustomProcess(Process):
    def __init__(self, limit):
        super().__init__()
        self._limit = limit

    def run(self):
        for i in range(self._limit):
            print(f"From CustomProcess: {i}")

if __name__ == "__main__":
    cpr = CustomProcess(5)
    cpr.start()
```

```
From CustomProcess: 0
From CustomProcess: 1
From CustomProcess: 2
From CustomProcess: 3
From CustomProcess: 4
```

Рисунок 4 – Результат выполнения программы

5. Пример 5: Принудительное завершение выполнения процесса с помощью метода terminate().

Листинг программы:

```

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

from multiprocessing import Process
from time import sleep


def func():
    counter = 0
    while True:
        print(f"counter = {counter}")
        counter += 1
        sleep(0.1)

if __name__ == "__main__":
    proc = Process(target=func)
    proc.start()
    sleep(0.7)
    proc.terminate()

```

```

counter = 0
counter = 1
counter = 2
counter = 3
counter = 4
counter = 5
counter = 6

```

Рисунок 5 – Результат выполнения программы

6. Пример 6: Создание демон-процесса, который автоматически завершится при завершении главного процесса.

Листинг программы:

```

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

from multiprocessing import Process
from time import sleep


def func(name):
    counter = 0

```

```

while True:
    print(f"proc {name}, counter = {counter}")
    counter += 1
    sleep(0.1)

if __name__ == "__main__":
    proc1 = Process(target=func, args=("proc1",), daemon=True)
    proc2 = Process(target=func, args=("proc2",))
    proc2.daemon = True
    proc1.start()
    proc2.start()
    sleep(0.3)

```

```

-> ./loop_process.py
proc proc1, counter = 0
proc proc2, counter = 0
proc proc1, counter = 1
proc proc2, counter = 1
proc proc1, counter = 2
proc proc2, counter = 2

```

Рисунок 6 – Результат выполнения программы

7. Индивидуальное задание: С использованием многопроцессорности для заданного значения x найти сумму ряда S с точностью члена ряда по абсолютному значению $\varepsilon = 10^{-7}$ и произвести сравнение полученной суммы с контрольным значением функции y для двух бесконечных рядов.

$$S = \sum_{n=0}^{\infty} (-1)^n \frac{x^n}{2^{n+1}} = \frac{1}{2} - \frac{x}{4} + \frac{x^2}{8} - \dots;$$

$$x = 1, 2; y = \frac{1}{2+x}.$$

Рисунок 7 – Исходный ряд, значение переменной x , аналитическая формула
ряда y

Листинг основной программы:

```

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

from multiprocessing import Process, Queue, cpu_count
from math import ceil

```

```

class SeriesCalculator:

    def __init__(self, x: float = 1.2, eps: float = 10**-7) -> None:
        self.x: float = x
        self.eps: float = eps

        self.series_sum: float | None = None
        self.analytic_value: float = self._analytic_function()

    def _series_term(self, n: int) -> float:
        return ((-1) ** n) * (self.x**n) / (2 ** (n + 1))

    def _analytic_function(self) -> float:
        return 1 / (2 + self.x)

    def series_sum_sequential(self) -> float:
        s = 0.0
        n = 0
        while True:
            term = self._series_term(n)
            if abs(term) < self.eps:
                break
            s += term
            n += 1
        self.series_sum = s
        return s

    @staticmethod
    def _part_sum(x: float, n_start: int, n_end: int, eps: float, queue: Queue) -> None:
        s: float = 0.0

        for n in range(n_start, n_end):
            term = ((-1) ** n) * (x ** n) / (2 ** (n + 1))
            if abs(term) < eps:
                break
            s += term
        queue.put(s)

    def multiprocess_series(self) -> None:
        n_proc: int = cpu_count()
        queue: Queue = Queue()

        N: int = 1000
        chunk_size: int = ceil(N/n_proc)

```

```

processes: list[Process] = []

for i in range(n_proc):
    n_start: int = i * chunk_size
    n_end: int = (i+1)*chunk_size
    p = Process(target=self._part_sum, args=(self.x, n_start, n_end,
self.eps, queue))
    processes.append(p)
    p.start()

total_sum: float = 0.0
for _ in processes:
    total_sum += queue.get()

for p in processes:
    p.join()

self.series_sum = total_sum
return total_sum

def absolute_error(self) -> float:

    if self.series_sum is None:
        raise ValueError("Сначала нужно вызвать multiprocess_series()")
    return abs(self.series_sum - self.analytic_value)

def __str__(self) -> str:
    return (
        f"Ряд: S = Σ [(-1)^n * x^n / 2^(n+1)], n = 0..∞\n"
        f"x = {self.x}, eps = {self.eps}\n"
        f"Сумма ряда S = {self.series_sum}\n"
        f"Аналитическое значение y = {self.analytic_value}\n"
        f"Абсолютная погрешность |S - y| = {self.absolute_error()}\n"
        f"self.series_sum else 'N/A'\n"
    )

```

Листинг вызова программы:

```

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

from multiprocessing_sum_series import SeriesCalculator
import time

if __name__ == "__main__":
    calculator = SeriesCalculator(x=1.2, eps=1e-7)

```

```

start_time = time.time()
calculator.series_sum_sequential()
sequential_time = time.time() - start_time
print(f"Последовательное вычисление: {sequential_time:.6f} с")
print(calculator)

start_time = time.time()
calculator.multiprocess_series()
multiprocess_time = time.time() - start_time
print(f"\nМногопроцессное вычисление: {multiprocess_time:.6f} с")
print(calculator)

```

Последовательное вычисление: 0.000027 с
 Ряд: $S = \sum [(-1)^n * x^n / 2^{(n+1)}]$, $n = 0..\infty$
 $x = 1.2$, $\text{eps} = 1e-07$
 Сумма ряда $S = 0.31250004145136007$
 Аналитическое значение $y = 0.3125$
 Абсолютная погрешность $|S - y| = 4.145136006661332e-08$

 Многопроцессное вычисление: 0.272765 с
 Ряд: $S = \sum [(-1)^n * x^n / 2^{(n+1)}]$, $n = 0..\infty$
 $x = 1.2$, $\text{eps} = 1e-07$
 Сумма ряда $S = 0.31250004145136007$
 Аналитическое значение $y = 0.3125$
 Абсолютная погрешность $|S - y| = 4.145136006661332e-08$

Рисунок 8 – Результат выполнения программы

Вывод: в ходе выполнения работы были получены навыки по написанию многозадачных приложений на языке программирования Python версии 3.x.