

Języki skryptowe

dokumentacja projektu Płetwonurek

Wojciech Królik, gr. 3E

2 stycznia 2022

Część I

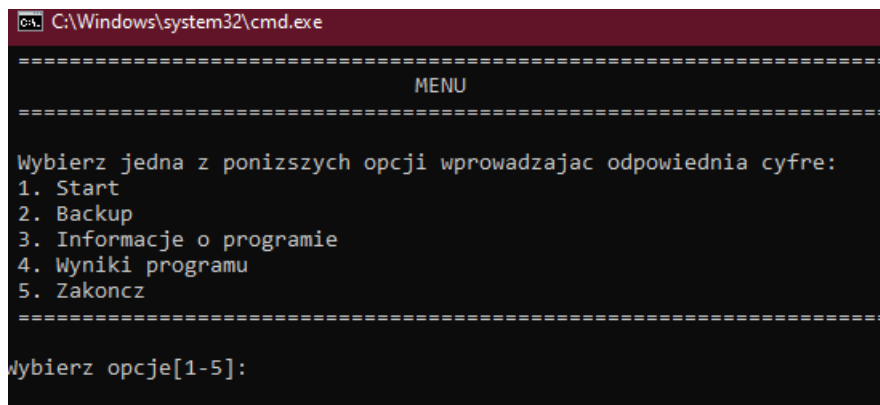
Opis programu

V OI, Etap II - Zadanie Płetwonurek

Płetwonurek do nurkowania używa butli, w której są dwa zbiorniki: z tlenem i z azotem. W zależności od czasu przebywania pod wodą i głębokości nurek potrzebuje różnych ilości tlenu i azotu. Płetwonurek ma do dyspozycji pewną liczbę butli. Każda butla charakteryzuje się wagą oraz objętością zawartego w niej tlenu i azotu. Do wykonania zadania nurek potrzebuje określonych ilości tlenu i azotu. Jaka jest najmniejsza sumaryczna waga butli, które nurek musi zabrać ze sobą, żeby mógł wykonać zadanie?

Instrukcja obsługi

Program jest obsługiwany z poziomu wiersza poleceń, w tym celu należy uruchomić plik "menu.bat". W nim znajdują się dalsze instrukcje co do programu.



```
C:\Windows\system32\cmd.exe

=====
                        MENU
=====

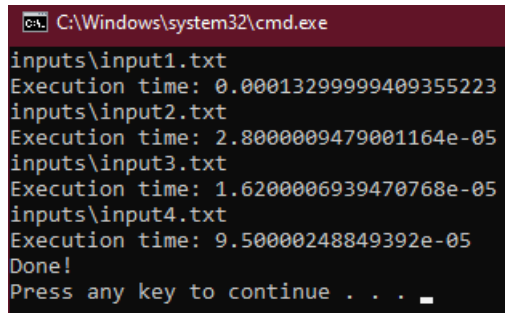
Wybierz jedna z ponizszych opcji wprowadzajac odpowiednia cyfre:
1. Start
2. Backup
3. Informacje o programie
4. Wyniki programu
5. Zakoncz
=====

Wybierz opcje[1-5]:
```

Rysunek 1: Instrukcje obsługi programu.

1. Start

Wybierając opcję pierwszą uruchomi nam się nasz główny program, czyli skrypt pythonowy zapisany w pliku main.py. Uruchamiając skrypt zostaną wyczyszczone pliki tekstowe z mierzonym czasem oraz z outputem, a następnie zacznie się wykonywanie programu, czas będzie mierzony przy uruchamianiu skryptu zapisanego w pliku ScubaDiver dla każdego osobnego pliku wejściowego i będzie zapisywany odpowiednio do plików tekstowych z outputem, to samo dotyczy mierzonego czasu.

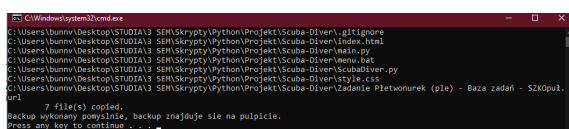


```
C:\Windows\system32\cmd.exe
inputs\input1.txt
Execution time: 0.00013299999409355223
inputs\input2.txt
Execution time: 2.8000009479001164e-05
inputs\input3.txt
Execution time: 1.6200006939470768e-05
inputs\input4.txt
Execution time: 9.50000248849392e-05
Done!
Press any key to continue . . .
```

Rysunek 2: Wykonywanie się programu i zapisywanie potrzebnych informacji i plików.

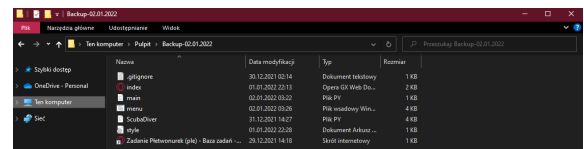
2. Backup

Wybierając opcję drugą - Backup, skrypt utworzy folder na pulpicie z datą wykonywania backupu, a następnie przekopiuje wszystkie pliki projektu do tego oto folderu. Po pomyślnie wykonanym backupie skrypt powinien nam wypisać, że wszystko przebiegło pomyślnie. Poniżej, jak powinien ten proces wyglądać.



```
C:\Windows\system32\cmd.exe
C:\Users\bunni\Desktop\STUDIA\3 SEM\Skrypty\Python\Projekt\Scuba-Diver\gitignore
C:\Users\bunni\Desktop\STUDIA\3 SEM\Skrypty\Python\Projekt\Scuba-Diver\index.html
C:\Users\bunni\Desktop\STUDIA\3 SEM\Skrypty\Python\Projekt\Scuba-Diver\main.py
C:\Users\bunni\Desktop\STUDIA\3 SEM\Skrypty\Python\Projekt\Scuba-Diver\menu.bat
C:\Users\bunni\Desktop\STUDIA\3 SEM\Skrypty\Python\Projekt\Scuba-Diver\ScubaDiver.py
C:\Users\bunni\Desktop\STUDIA\3 SEM\Skrypty\Python\Projekt\Scuba-Diver\style.css
C:\Users\bunni\Desktop\STUDIA\3 SEM\Skrypty\Python\Projekt\Scuba-Diver\Zadanie Pletwonurek (pie) - Baza zadani - SZKOpul.
git
7 file(s) copied.
Backup wykonany pomyślnie, backup znajduje się na pulpicie.
Press any key to continue . . .
```

(a) Wykonanie kopii zapasowej.



(b) Utworzony folder *Backup* z dzisiejszą datą.

Rysunek 3: Działanie funkcji Backup'u.

3. Informacje

Wybierając opcję trzecią - Informacje o programie, program pokazuje nam wszystkie informacje o projekcie - o tym jak przechowywane są dane wejściowe oraz wyjściowe, jest tu również informacja o autorze jak i ogólne założenia, wymagania projektu.

```
C:\Windows\system32\cmd.exe

=====
INFORMACJE O PROGRAMIE I JEGO DZIAŁANIE
=====

Zadaniem programu Scuba-Diver jest dobranie takich zestawow dla ple-
-twonurka, by spelnialy one wszystkie wymagania a przy tym zeby ich
waga byla jak najmniejsza.

Kazdy zestaw sklada sie z nastepujacych elementow:
- tlenu
- azotu
- wagi

W pliku wejsciowym mamy zadane wymagania czyli:
- minimalna ilosc tlenu
- minimalna ilosc azotu

Program na podstawie tych wymagan dobierze takie zestawy by spelni-
-ly one wszystkie wymagania i wazyly przy tym jak najmniej.
=====
DANE WEJSCIOWE
=====

Dane wejsciowe sa zapisywane w folderze Inputs, dla kazdego zestawu
mamy osobny plik wejsciowy input[x].txt, gdzie x to numer zestawu.
Przykladowy zestaw wyglada nastepujaco:

5 60      #wymagana ilosc tlenu, wymagana ilosc azotu
5         #zadeklarowana liczba zestawow
3 36 120  #ilosc tlenu, ilosc azotu, waga zestawu
10 25 129
5 50 250
1 45 130
4 20 119

W ten sam sposob zapisany jest kazdy inny plik wejsciowy.

=====
DANE WEJSCIOWE
=====

Dane wyjsciowe sa zapisywane w folderze Outputs, dla kazdego zestawu
mamy osobny plik wyjsciowy output[x].txt, gdzie x to numer zestawu.

W kazdym pliku jest zapisana liczba, ktora jest minimalna z wag zestawow.

=====
Autor programu: Wojciech Krolik, Informatyka II rok, grupa: 3E
```

Rysunek 4: Informacje o projekcie.

4. Wyniki programu

Wybierając opcję czwartą, zostaniemy przeniesieni do strony internetowej stworzonej na potrzeby projektu. Są na niej zapisane informacje statystyczne, takie jak dane wejściowe, dane wyjściowe, oraz dodatek w postaci czasu wykonaniu skryptu pythona dla każdego inputu. Strona została zaprojektowana w języku znacznikowym HTML5, oraz użyte zostały drobne style, przy użyciu CSS - głównie do przedstawienia tabeli. Została tam dodatkowo zawarta informacja o autorze.

Scuba Diver Tables

Input data	Output data	Execution time
5 60 5 3 36 120 10 25 129 5 50 250 1 45 130 4 20 119	249	0.00013299999409355223
6 50 4 5 36 200 3 20 129 5 50 250 2 30 110	379	2.8000009479001164e-05
8 70 3 5 56 200 5 10 250 3 20 110	310	1.6200006939470768e-05
6 70 6 3 36 120 10 25 129 5 50 250 1 45 130 4 20 119 1 10 150	370	9.50000248849392e-05

Wojciech Królik, 2022, Języki Skryptowe - Projekt 1

Rysunek 5: Strona internetowa.

5. Zakończ

Ostatnią z dostępnych opcji jest wyjście z programu, jej użycie spowoduje wyjście z wiersza poleceń.

Dodatkowe informacje

Dla poprawnego działania programu potrzebny jest zainstalowany Python w wersji 3.0 \geq .

Część II

Opis działania

Cały algorytm rozwiązania problemu płetwonurka został opracowany na klasie ScubaDiver. Jako wejście do konstruktora klasy wczytywane są kolejne liczby z pliku tekstowego, każda liczba odpowiadać będzie innemu atrybutowi, mamy więc następujące atrybuty:

- Potrzebny tlen - *oxygen needed*,
- Potrzebny azot - *nitrogen needed*,
- Wszystkie zestawy - *kits*.

Kolejno atrybuty są następującymi typami danych: int, int, list. W konstruktorze, dla którego jako argument podajemy plik każda z wartości jest sprawdzana zgodnie z logiką czyli:

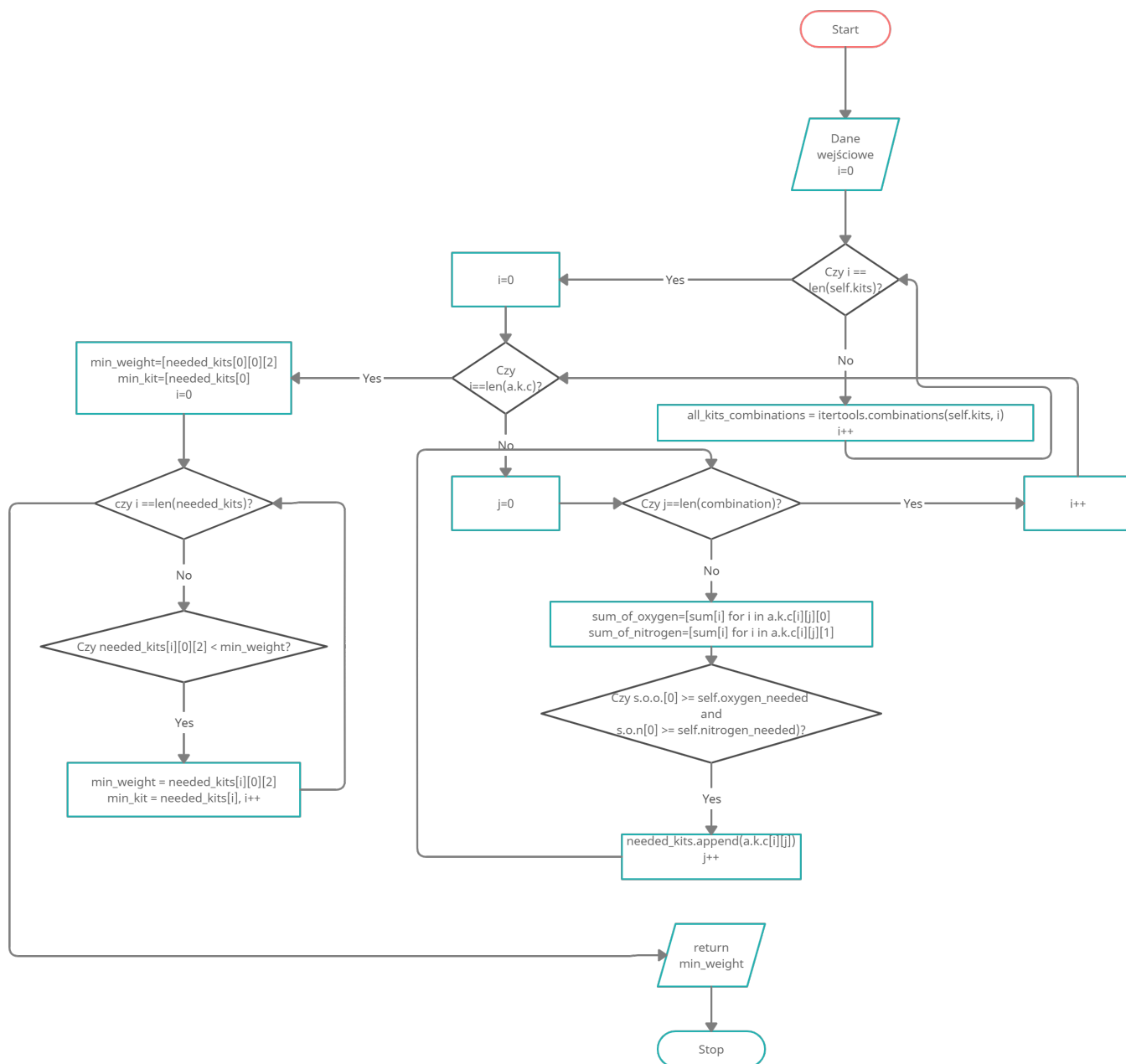
- Tlen nie może być większy od 20 i mniejszy od 0 i musi być liczbą całkowitą,
- Azot nie może być większy od 78 i mniejszy o 0 i musi być liczbą całkowitą,
- Zestaw składa się z tlenu, azotu oraz wagi, która również jest liczbą całkowitą

Po przyjęciu danych przez konstruktor, by wydobyć z zestawów najmniejszego spełniającego warunki zestawu algorytm będzie działał następująco:

Na samym początku pobierzemy wszystkie możliwe kombinacje złożonych zestawów - spełniających i niespełniających warunki, następnie zwrócimy ją i przypiszemy do zmiennej w postaci listy krotek list.

Następnym krokiem będzie sprawdzenie, w których zestawach zawartość tlenu oraz azotu jest zgodna z wymaganiami zadania. Wszystkie zestawy niespełniających tych wymagań zostaną pominięte i zwrócona zostanie nam lista wszystkich zestawów, które je spełniają, tym razem w postaci listy list(zestawów).

Ostatnim z kroków będzie sprawdzenie, w którym z zestawów łączna waga jest najmniejsza. Sprawdzenie polega na prostym porównaniu wartości. Minimalna waga zostaje zwrócona. Poniżej zostanie zaprezentowany schemat blokowy przedstawionego wyżej algorytmu.



Rysunek 6: Schemat blokowy przedstawionego powyżej algorytmu.

Algorytm

Pseudokod tworzymy w \LaTeX . Przykład:

Definicja 1:

Data: Dane wejściowe `input[i].txt`

all – kits – combinations =

`[itertools.combinations(self.kits, i) for i in range(len(self.kits))]`

for *combination* in *all – kits – combinations* **do**

for *list – of – kits* in *combination* **do**

sum – of – oxygen = suma tlenu wszystkich zestawów w zestawie

sum – of – nitrogen = suma azotu wszystkich zestawów w zestawie

if *sum – of – oxygen* \geq *oxygen – needed* and *sum – of – nitrogen* \geq *nitrogen – needed* **then**

needed – kits.append(*list-of-kits*)**end**

end

end

min-weight = pierwsza waga z listy *needed-kits*

min-kit = pierwszy zestaw z listy

for *kit* in *needed – kits* **do**

if *aktualna waga kit* $<$ *min – weight* **then**

min – weight = *aktualna waga kit*

min – kit = *aktualny kit*

end

end

Result: return *min – weight*;

Algorithm 1: Algorytm szukania najbliższego zestawu dla pletwonurka.

Plik main, statystyka oraz odczyt i zapis do pliku.

Żeby skrypt pythonowy mógł poprawnie działać, potrzebne i żeby została zachowana poprawna struktura projektu, projekt został rozdzielony na dwa pliki, jeden zawierający klasę ScubaDiver, której algorytm został już omówiony oraz drugi plik - *main.py* w którym zawarte zostało, wczytywanie z pliku danych wejścia oraz mierzenie czasu w celach statystycznych. Zostały tam dodane biblioteki glob - służąca przeglądaniu plików oraz timeit - mierząca czas pomiędzy uruchomieniem a zakończeniem działania procesów algorytmu.

Program początkowo otwiera wszystkie pliki z outputem z poprzednich uruchomień i czyści je, następnie wpisuje wyniki programu do plików z odpowiadającymi im numerami iteracji plików.

```
1 import glob
2 from ScubaDiver import ScubaDiver
3
4 from timeit import default_timer as timer
5
6 inputs = glob.glob("inputs/*.txt")
7 for i, file in enumerate(inputs):
8     with open(f"outputs/output{i+1}.txt", "w") as f:
9         if f.tell() == 0:
10             f.write("")
11     with open(f"clocks/time{i+1}.txt", "w") as f:
12         if f.tell() == 0:
13             f.write("")
14
15 for i, file in enumerate(inputs):
16     print(file)
17     projekt = ScubaDiver(file)
18     start = timer()
19     weight = projekt.get_minimal_weight_needed_kit()
20     execution_time = timer() - start
21     print(f"Execution time: {execution_time}")
22     with open(f"outputs/output{i+1}.txt", "a") as f:
23         f.write(str(weight) + "\n")
24     with open(f"clocks/time{i+1}.txt", "a") as f:
25         f.write(str(execution_time) + "\n")
26
27 print("Done!")
```

Pliki wejścia:

Data: Dane wejściowe *input[i].txt*

Otwierane zostają pliki outputu oraz są odpowiednio czyszczone, to samo dotyczy plików z czasem. Otwierane zostają pliki inputu oraz wczytywane są kolejne linie z plików. Pomiedzy uruchomieniami programu włączany jest timer i jest liczony czas wykonania programu. Na samym końcu dostajemy wiadomość, że wszystko przebiegło pomyślnie.

Algorithm 2: Pseudokod dot. pliku main.

Testy

Dla sprawdzenia poprawności działania programu i porównania została stworzona prosta strona HTML z tabelą, do której automatycznie wpisują się: output, input, timer

Scuba Diver Tables

Input data	Output data	Execution time
5 60 5 3 36 120 10 25 129 5 50 250 1 45 130 4 20 119	249	0.00013299999409355223
6 50 4 5 36 200 3 20 129 5 50 250 2 30 110	379	2.8000009479001164e-05
8 70 3 5 56 200 5 10 250 3 20 110	310	1.6200006939470768e-05
6 70 6 3 36 120 10 25 129 5 50 250 1 45 130 4 20 119 1 10 150	370	9.50000248849392e-05

Wojciech Królik, 2022, Języki Skryptowe - Projekt 1

Rysunek 7: Tabela statystyczna w witrynie internetowej.

Każdy wynik zwraca prawdziwą wartość co dowodzi, że program działa pomyślnie.

Pelen kod Scuba-Diver.py

```
1 import itertools
2
3
4 class ScubaDiver:
5     """ScubaDiver class which represents a scuba diver who needs to find
6         the oxygen and nitrogen kits to fill his tank."""
7
8     def __init__(self, file) -> None:
9         """
10            Initialize the ScubaDiver object.
11            Parameters:
12                file: The file to read.
13            """
14
15        with open(file) as f:
16            line = f.readline()
17            self.oxygen_needed = line.split(" ")[0]
18            try:
19                self.oxygen_needed = int(self.oxygen_needed)
20                if self.oxygen_needed < 0:
21                    raise ValueError("Oxygen needed must be positive")
22                if self.oxygen_needed > 21:
23                    raise ValueError("Oxygen needed must be less than 22")
24            except ValueError:
25                raise ValueError("Oxygen must be an integer.")
26
27        self.nitrogen_needed = line.split(" ")[1]
28        try:
29            self.nitrogen_needed = int(self.nitrogen_needed)
30            if self.nitrogen_needed < 0:
31                raise ValueError("Nitrogen needed must be positive")
32            if self.nitrogen_needed > 79:
33                raise ValueError("Nitrogen needed must be less than 80")
34        except ValueError:
35            raise ValueError("Nitrogen must be an integer.")
36
37        self.number_of_kits = f.readline()
38        try:
39            self.number_of_kits = int(self.number_of_kits)
40            if self.number_of_kits < 0:
41                raise ValueError("Number of kits must be positive.")
42            if self.number_of_kits > 1000:
43                raise ValueError("Number of kits must be less than 1000.")
44        except ValueError:
45            raise ValueError("Number of kits must be an integer.")
46
47        self.kits = [f.readline().split(" ") for i in range(self.number_of_kits)]
```

```

48
49         for i in range(self.number_of_kits):
50             self.kits[i][-1] = self.kits[i][-1].strip()
51
52         try:
53             self.kits = [[int(i) for i in kit] for kit in self.kits]
54         except ValueError:
55             raise ValueError("Kits must be integers.")
56
57     def get_all_kits_combinations(self):
58         """
59         Get all the combinations of kits.
60         """
61         all_kits_combinations = []
62         for i in range(len(self.kits)):
63             combination_kit = itertools.combinations(self.kits, i)
64             combination_list = list(combination_kit)
65             all_kits_combinations.append(combination_list)
66         return all_kits_combinations
67
68     def get_needed_kits_to_fit_tank(self):
69         """
70         Get the kits needed to fill the tank.
71         """
72         needed_kits = []
73         all_kits_combinations = self.get_all_kits_combinations()
74         for combination in all_kits_combinations:
75             for list_of_kits in combination:
76                 sum_of_oxygen = [sum(i[0] for i in list_of_kits)]
77                 sum_of_nitrogen = [sum(i[1] for i in list_of_kits)]
78                 if (
79                     sum_of_oxygen[0] >= self.oxygen_needed
80                     and sum_of_nitrogen[0] >= self.nitrogen_needed
81                 ):
82                     needed_kits.append(list_of_kits)
83         return needed_kits
84
85     def get_minimal_weight_needed_kit(self):
86         """
87         Get the minimal weight needed kit.
88         """
89         needed_kits = self.get_needed_kits_to_fit_tank()
90         minimal_weight = needed_kits[0][0][2]
91         minimal_weight_kit = needed_kits[0]
92         for kit in needed_kits:
93             if kit[0][2] < minimal_weight:
94                 minimal_weight = kit[0][2]
95                 minimal_weight_kit = kit
96         minimal_weight = sum(i[2] for i in minimal_weight_kit)
97         return minimal_weight

```

Peŝen kod main.py

```
1 import glob
2 from ScubaDiver import ScubaDiver
3
4 from timeit import default_timer as timer
5
6 inputs = glob.glob("inputs/*.txt")
7 for i, file in enumerate(inputs):
8     with open(f"outputs/output{i+1}.txt", "w") as f:
9         if f.tell() == 0:
10             f.write("")
11     with open(f"clocks/time{i+1}.txt", "w") as f:
12         if f.tell() == 0:
13             f.write("")
14
15 for i, file in enumerate(inputs):
16     print(file)
17     projekt = ScubaDiver(file)
18     start = timer()
19     weight = projekt.get_minimal_weight_needed_kit()
20     execution_time = timer() - start
21     print(f"Execution time: {execution_time}")
22     with open(f"outputs/output{i+1}.txt", "a") as f:
23         f.write(str(weight) + "\n")
24     with open(f"clocks/time{i+1}.txt", "a") as f:
25         f.write(str(execution_time) + "\n")
26
27 print("Done!")
```

Pelen kod index.html

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <meta charset='utf-8'>
6     <title>Scuba Diver Tables</title>
7     <meta name='viewport' content='width=device-width, initial-scale=1'>
8     <link rel='stylesheet' type='text/css' media='screen' href='style.
      css'>
9 </head>
10
11 <body>
12     <header>
13         <h1>Scuba Diver Tables</h1>
14     </header>
15     <table>
16         <tr>
17             <th>Input data</th>
18             <th>Output data</th>
19             <th>Execution time</th>
20         </tr>
21         <tr>
22             <td>
23                 <object data="inputs/input1.txt"></object>
24             </td>
25             <td>
26                 <object data="outputs/output1.txt"></object>
27             </td>
28             <td>
29                 <object data="clocks/time1.txt"></object>
30             </td>
31         </tr>
32         <tr>
33             <td>
34                 <object data="inputs/input2.txt"></object>
35             </td>
36             <td>
37                 <object data="outputs/output2.txt"></object>
38             </td>
39             <td>
40                 <object data="clocks/time2.txt"></object>
41             </td>
42         </tr>
43         <tr>
44             <td>
45                 <object data="inputs/input3.txt"></object>
46             </td>
47             <td>
48                 <object data="outputs/output3.txt"></object>
49             </td>
50             <td>
51                 <object data="clocks/time3.txt"></object>
```

```

52         </td>
53     </tr>
54     <tr>
55         <td>
56             <object data="inputs/input4.txt"></object>
57         </td>
58         <td>
59             <object data="outputs/output4.txt"></object>
60         </td>
61         <td>
62             <object data="clocks/time4.txt"></object>
63         </td>
64     </tr>
65 </table>
66
67 <footer>
68     <p>
69         Wojciech Krupa, 2022, Języki Skryptowe - Projekt 1
70     </p>
71 </footer>
72 </body>
73
74 </html>

```

Pelen kod style.css

```
1
2 table {
3     border-collapse: collapse;
4     border-spacing: 0;
5     width: auto;
6     border: 1px solid rgb(0, 0, 0);
7     margin: auto;
8
9 }
10 td{
11     border: 1px solid rgb(0, 0, 0);
12     text-align: center;
13     padding: 10px;
14     margin: -10px;
15 }
16 tr {
17     margin: auto;
18 }
19 object{
20     margin: auto;
21     display: block;
22     margin: 0 auto;
23 }
24 body {
25     font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
26     font-size: 14px;
27     line-height: 20px;
28     color: #333333;
29     background-color: #ffffff;
30     margin: 0 auto;
31     padding: 0;
32     text-align: center;
33 }
34 footer{
35     margin-top: 7.4%;
36     text-align: center;
37 }
```

Pełen kod menu.bat

```
1 @echo off
2 cd %~dp0
3 :menu
4 cls
5 echo
6 echo
7 echo
8 echo.
9 echo Wybierz jedna z ponizszych opcji wprowadzajac odpowiednia cyfre:
10 echo 1. Start
11 echo 2. Backup
12 echo 3. Informacje o programie
13 echo 4. Wyniki programu
14 echo 5. Zakoncz
15 echo
16 echo.
17 set /p choice="Wybierz opcje[1-5]: "
18
19 IF %choice%==1 GOTO startt
20 IF %choice%==2 GOTO backup
21 IF %choice%==3 GOTO info
22 IF %choice%==4 GOTO html
23 IF %choice%==5 GOTO exit
24
25 echo.
26 echo Wybierz jedna z podanych opcji.
27 pause
28 GOTO :menu
29
30 :startt
31 cls
32 python main.py
33 pause
34 GOTO :menu
35
36 :backup
37 cls
38 cd C:\Users\bunnv\Desktop
39 mkdir Backup-%date%
40 copy "C:\Users\bunnv\Desktop\STUDIA\3 SEM\Skrypty\Python\Projekt\Scuba-
    Diver" C:\Users\bunnv\Desktop\Backup-%date%
41 echo Backup wykonany pomyslnie, backup znajduje sie na pulpicie.
42 pause
43 GOTO :menu
44
45 :info
46 cls
47 echo.
48 echo
```

```

=====
49 echo          INFORMACJE O PROGRAMIE I JEGO DZIAŁANIE
50 echo
=====
51 echo.
52 echo  Zadaniem programu Scuba-Diver jest dobranie takich zestawow dla
    ple-
53 echo  -twonurka, by spelnialy one wszystkie wymagania a przy tym zeby
    ich
54 echo  waga byla jak najmniejsza.
55 echo.
56 echo  Kazdy zestaw sklada sie z nastepujacych elementow:
57 echo  - tlenu
58 echo  - azotu
59 echo  - wagi
60 echo.
61 echo  W pliku wejsciowym mamy zadane wymagania czyli:
62 echo  - minimalna ilosc tlenu
63 echo  - minimalna ilosc azotu
64 echo.
65 echo  Program na podstawie tych wymagan dobierze takie zestawy by spelni
    -
66 echo  -ly one wszystkie wymagania i wazyly przy tym jak najmniej.
67 echo
=====
68 echo          DANE WEJSCIOWE
69 echo.
70 echo  Dane wejsciowe sa zapisywane w folderze Inputs, dla kazdego
    zestawu
71 echo  mamy osobny plik wejsciowy input[x].txt, gdzie x to numer zestawu
    .
72
73 echo  Przykladowy zestaw wyglada nastepujaco:
74 echo.
75 echo    5 60          #wymagana ilosc tlenu, wymagana ilosc azotu
76 echo    5            #zadeklarowana liczba zestawow
77 echo    3 36 120      #ilosc tlenu, ilosc azotu, waga zestawu
78 echo    10 25 129
79 echo    5 50 250
80 echo    1 45 130
81 echo    4 20 119
82 echo.
83 echo  W ten sam sposob zapisany jest kazdy inny plik wejsciowy.
84 echo.
85 echo.
86 echo
=====
87 echo          DANE WEJSCIOWE
88 echo  Dane wyjsciowe sa zapisywane w folderze Outputs, dla kazdego
    zestawu
89 echo  mamy osobny plik wyjsciowy output[x].txt, gdzie x to numer
    zestawu.
90 echo.
91 echo  W kazdym pliku jest zapisana liczba, ktora jest minimalna z wag
    zestawow.

```

```
92 echo .
93 echo .
94 echo
=====
95 echo  Autor programu: Wojciech Krolik, Informatyka II rok, grupa: 3E
96 echo .
97 echo .
98 pause
99 GOTO :menu
100
101 :html
102 cls
103 cd "C:\Users\bunnv\Desktop\STUDIA\3 SEM\Skrypty\Python\Projekt\Scuba-
    Diver"
104 start index.html
105 GOTO :menu
106
107 :exit
108 exit
```
