Politechnika Śląska

# Języki Skryptowe

Dokumentacja projektu PIONEK

Kacper Grabiec, grupa 2C

# Część I

## Opis programu

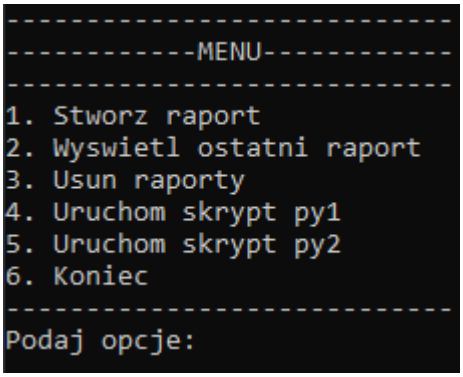
**PIONEK, XXV OI, etap I**

W punkcie (0, 0) nieskończonej kratki stoi pionek. Pionek ma *n* dozwolonych ruchów. Każdy z nich jest opisany za pomocą wektora o współrzędnych całkowitych. Pionek może każdy z ruchów wykonać co najwyżej raz, w dowolnej kolejności. Wektory opisujące ruchy mogą się powtarzać i wtedy pionek może wykorzystać każdy z nich.

Naszym celem jest dostać się pionkiem do punktu położonego możliwie najdalej od punktu początkowego (w odległości euklidesowej). Jak daleko może on dotrzeć?

## Instrukcja obsługi

W celu rozwiązania zadania należy uruchomić plik *bat.bat*. Po jego włączeniu ukaże się nam menu z kilkoma opcjami do wyboru.

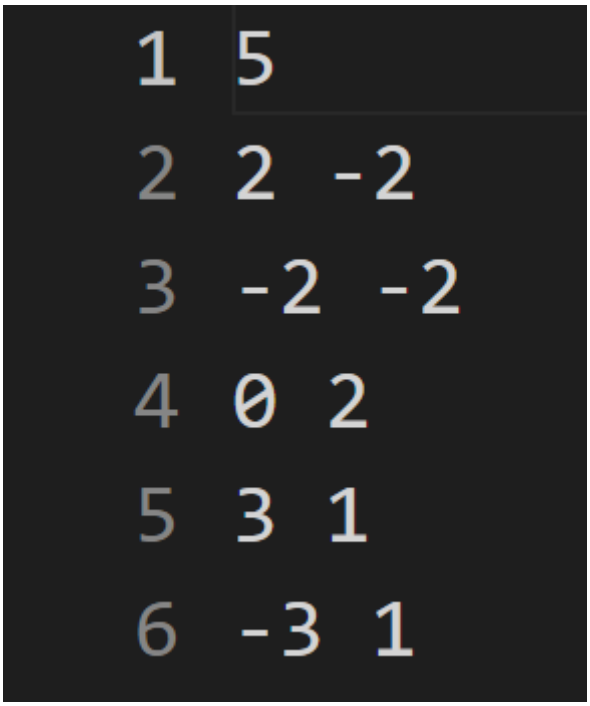



Program czeka, aż wybierzemy jedną z dostępnych opcji:

- **Stwórz raport** – program rozwiązuje problem dla zadanych danych wejściowych oraz tworzy raport na podstawie danych wyjściowych;
- **Wyświetl ostatni raport** – program wyświetla nam najnowszy plik *.html* zawierający gotowy raport;
- **Usuń raporty** – program usuwa wszystkie raporty znajdujące się w folderze */raporty*;
- **Uruchom skrypt py1.py** – program uruchamia tylko plik *py1.py*, odpowiedzialny za rozwiązanie zadania;
- **Uruchom skrypt py2.py** – program uruchamia tylko plik *py2.py*, odpowiedzialny za stworzenie raportu;
- **Koniec** – koniec działania programu;

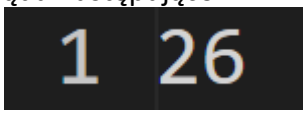
Po wykonaniu każdej opcji menu wyświetla się ponownie.

Każdy plik zawierający dane wejściowe (np. *in1.txt*) znajduje się w folderze */input_data*. Jego zawartość wygląda następująco:



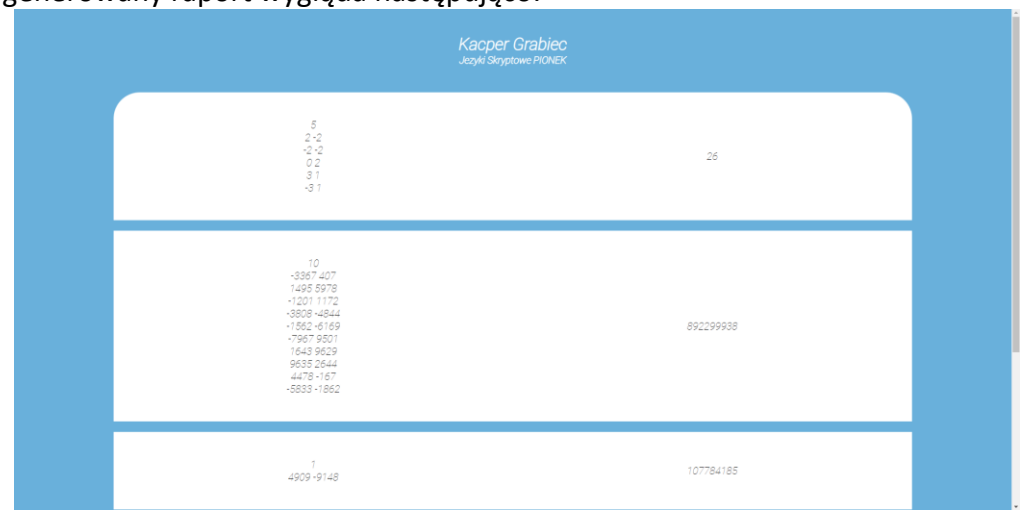

Pierwszy wiersz standardowego wejścia zawiera jedną dodatnią liczbę całkowitą *n* oznaczającą liczbę możliwych ruchów pionka. Każdy z kolejnych *n* wierszy zawiera dwie liczby całkowite *x*, *y* oddzielone pojedynczym odstępem i oznaczające wektor [*x, y*] opisujący możliwy ruch pionka.

Każdy plik zawierający dane wyjściowe (n. *out1.txt*) znajduje się w folderze */output_data*. Jego zawartość wygląda następująco:




Program wypisuje na standardowe wyjście liczbę całkowitą oznaczającą kwadrat odległości od punktu (0, 0) do najdalszego punktu, do którego może doskoczyć pionek.

Wygenerowany raport wygląda następująco:




Po lewej stronie wyświetlone zostają kolejne pliki zawierające dane wejściowe, po prawej dane wyjściowe.

# Część II

## Opis działania

Do rozwiązania zadania program wykorzystuje rekurencję. Po wczytaniu danych wejściowych program sprawdza wszystkie możliwości wywołując tą samą funkcję oraz sprawdzając, czy dany ruch nie został już wykorzystany. Jeśli wszystkie ruchy zostały wykorzystane program tworzy dane wyjściowe, tworzy raport oraz kończy działanie.

## Implementacja

```
28  def solve():
29      for i in range(0, n):
30          skip = False
31          vector = [0, 0]
32          global max_distance
33          for j in range(0, len(indexes)):
34              if i == indexes[j]:
35                  skip = True
36                  break
37          if skip:
38              continue
39          else:
40              indexes.append(i)
41              for j in range(0, len(indexes)):
42                  vector = sum_vectors(vector, moves[indexes[j]])
43              results.append(vector)
44              distance = calc_sqr_vec_len(results[-1])
45              if distance > max_distance:
46                  max_distance = distance
47              while len(results) > 0:
48                  results.pop()
49              solve()
50              indexes.pop()
```

Funkcja *solve()* jest najważniejszą częścią w całym programie. Za pomocą tablicy *indexes[]* sprawdza, czy dane ruchy nie zostały już wykorzystane. Następnie wywołuje inne funkcje, które np. obliczają dystans. Na końcu porównuje wynik z największym dystansem *max_distance* i usuwa wykorzystane już dane.

```
4   def load_data(filename: str):
5       file = open('./input_data/' + filename, 'r')
6       data = []
7       for line in file:
8           line = line.strip()
9           data.append([int(value) for value in line.split(' ')])
10      file.close()
11      return data
```

Funkcja *load_data()* służy do wczytania plików z danymi wejściowymi. Funkcja czyta plik i zwraca zawartość w postaci tablicy.

```
14  def print_data(filename: str, data: str):
15      file = open('./output_data/' + filename, 'w')
16      file.write(data)
17      file.close()
```

Funkcja *print_data()* tworzy plik z danymi wyjściowymi.

```
20  def calc_sqr_vec_len(vector):
21      return pow(vector[0], 2) + pow(vector[1], 2)
22
23
24  def sum_vectors(v1, v2):
25      return [v1[0] + v2[0], v1[1] + v2[1]]
```

Funkcja *calc_sqr_vec_len()* oblicza nam długość wektora natomiast *sum_vectors()* zwraca sumę wektorów.

```
34  def print_data():
35      time = datetime.now()
36      time = time.strftime("%H-%M-%S")
37      today = date.today()
38      today = today.strftime("%d-%m-%Y")
39      file = open(f'./raporty/{today}_{time}_raport.html', 'w')
40      file.write(f'''
41      <!DOCTYPE HTML>
42      <html lang="pl-PL">
43          <head>
44              <meta charset="utf-8">
45              <title>{today}_{time}_Raport</title>
46              <meta name="viewport" content="width=device-width, initial-scale=1">
47
48              <link rel="stylesheet" href="../style.css">
49          </head>
50
51          <body>
52              <h1>Kacper Grabiec</h1>
53              <h3>Jezyki Skryptowe PIONEK</h3>
54              {generate_template()}
55          </body>
56      </html>
57      ''')
58      file.close()
```

Funkcja *print_data()* pobiera obecny czas oraz datę, a następnie tworzy raport. Wypisuje do pliku podstawowy wzorzec składni *HTML*. W sekcji *<body>* wywołuje funkcję *generate_template()* odpowiedzialną za odpowiednie umiejscowienie danych wejściowych i wyjściowych w raporcie.

```
16  def generate_template():
17      template = ''
18      for i in range(0, len(input_files)):
19          template += '''
20          <div class="line">
21              <div class="data">
22          '''
23          for j in input_files[i]:
24              template += f'''
25                  <div class="input">{j}</div>
26              '''
27          template += f'''
28              </div>
29              <div class="data">{output_files[i][0]}</div>
30          </div>'''
31      return template
```

Funkcja *generate_template()* tworzy osobne elementy HTML dla każdych danych wejściowych oraz wyjściowych.

## Pełen kod aplikacji

https://github.com/KacperGrabiec/JezykiSkryptoweProjekt