Python - losowe zagadnienia

Kamil Musikowski

25 stycznia 2022

Streszczenie

Projekt na zaliczenie warsztatu programisty. Wybrałem temat Python - losowe zagadnienia, dlatego iż że nie chciałem przepisywać gotowej książki, ani tutorialu pythona, co ktoś już pewnie zrobił.

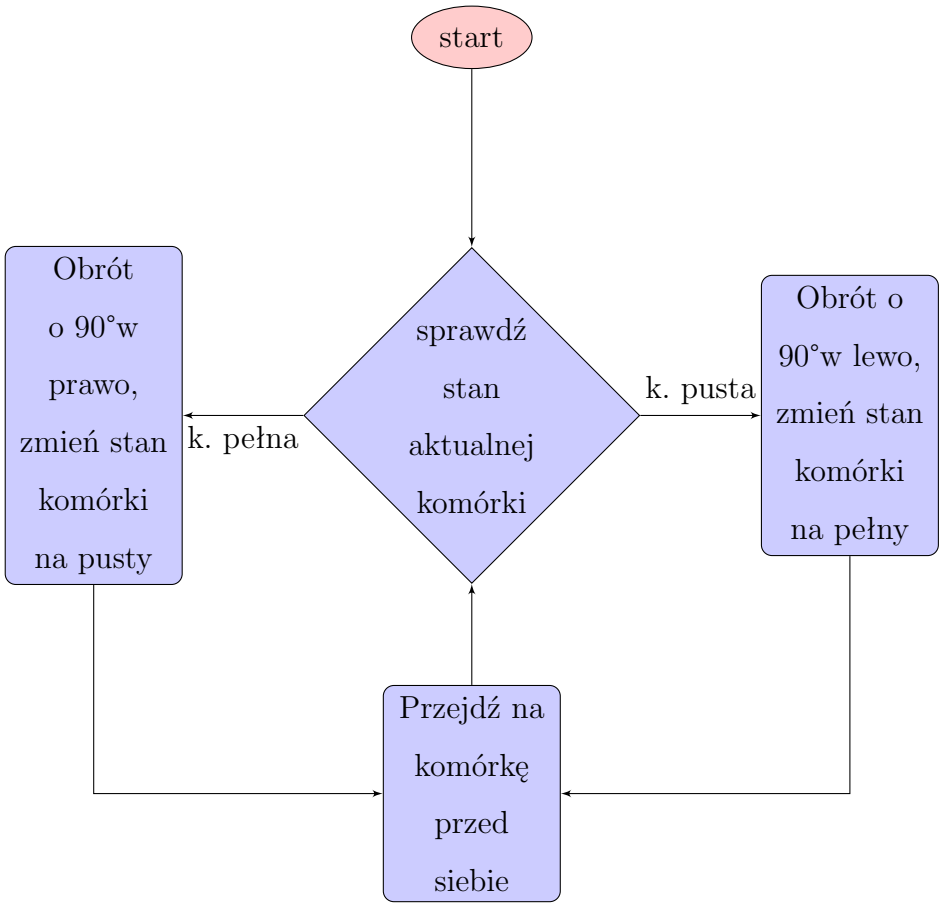
Spis treści

1 Mrówka langtona

Jest to automat komórkowy jak i również Maszyna Turinga działająca na następujących zasadach:

- Na mapie składającej się z pełnych i pustych komórek pojawia się mrówka
- Jeśli mrówka znajduje się na polu pustym, to obraca się w lewo o 90°, po czym przechodzi na następną komórkę
- Jeśli znajdzie się na polu pełnym, obraca się w prawo o 90°i przechodzi na następną komórkę

Schemat blokowy:



```python
class Ant:
    def __init__(self, y, x, direction):
        self.y = y
        self.x = x
        self.direction = direction

    def create_area(self, size):
        return [['[ ]' for _ in range(size)] for _ in range(
           size)]
    def change_area(self):
        if area[self.y][self.x] == '[ ]':
            area[self.y][self.x] = '[#]'
        else:
            area[self.y][self.x] = '[ ]'

    def change_location(self):
        if self.direction == 1:
            self.y -= 1
        elif self.direction == 2:
            self.x += 1
        elif self.direction == 3:
            self.y += 1
        elif self.direction == 4:
            self.x -= 1

    def choose_turn_angle(self):
        if area[self.y][self.x] == '[ ]':
            return True
        else:
            return False

    def right_turn(self):
        self.direction += 1
        if self.direction == 5:
            self.direction = 1
    def left_turn(self):
        self.direction -= 1
        if self.direction == 0:
            self.direction = 4

    def show_area(self):
        for row in area:
            print(" ".join(map(str, row)))  # analyze

ant = Ant(7, 7, 1)  #startowe y, x, kierunek
area = ant.create_area(15) #wielkosc mapy
for reloads in range(400): #ilosc ruchow
    ant.change_area()
    ant.change_location()
    if ant.choose_turn_angle():
        ant.right_turn()
    else:
        ant.left_turn()
ant.show_area()
```

Oto jak owy program może wyglądać, na wypadek gdyby komuś nie chciało się go włączać:

Przykładowy gif dla pierwszych 200 kroków

Bardziej zaawansowana wersja

Przykładowy wynik po 11000 kroków[4]:

2 Operacje na listach

2.1 Slicing

Przykładowe użycia slicingu.

Slicing może być używany na pętlach i tuplach.

```
>>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> pierwsze_piec = numbers[0:5]
>>> pierwsze_piec[2] = 30
>>> pierwsze_piec
[1, 2, 30, 4, 5]
>>> numbers
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Wybieranie środkowych elementów listy:

```
>>> numbers = [10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> numbers[1:-1]
[20, 30, 40, 50, 60, 70, 80]
>>> numbers[-3:8]
[70, 80]
>>> numbers[-5:-1]
[50, 60, 70, 80]
```

Przypisywanie wartości zmiennym za pomocą slicingu:

```
>>> numbers = [10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> numbers[:4] = [1,2,3,4]
>>> numbers
[1, 2, 3, 4, 50, 60, 70, 80, 90]
```

Przypisywanie wartości i zmiana wielkości listy:

```
>>> numbers = [10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> numbers[:4] = [1,2,3,4,5,6,7]
>>> numbers
[1, 2, 3, 4, 5, 6, 7, 50, 60, 70, 80, 90]
```

Zastępywanie co n-tego elementu:

```
>>> numbers = [10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> numbers[::2] = [1,1,1,1,1]
>>> numbers
[1, 20, 1, 40, 1, 60, 1, 80, 1]
```

Zastępywanie co n-tego elementu, ale od końca:

```
>>> numbers = [10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> numbers[::-2] = [1,2,3,4,5]
>>> numbers
[5, 20, 4, 40, 3, 60, 2, 80, 1]
```

Odwracanie listy:

```
>>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> numbers[::-1]
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

3 Przydatne funkcje

3.1 Funkcja lambda:

Funkcję Lambda, inaczej można nazwać funkcją anonimową lub literałem funkcyjnym. Funkcja ta nie jest czymś, czego nie można osiągnąć za pomocą standardowych sposobów. Używa jej się przede wszystkim ze względu na wygodę i czytelność. Jest często używana raz, lub niewielką ilość razy, jest szczególnie przydatna w dziedzinie analizy danych, chociażby jako pomoc przy użyciu funkcji map(), filter(), reduce().

Użycie funkcji lambda wygląda następująco:

```
1 multiply = lambda x, y: x*y
2 multiply(3, 5)
```

I jest równoznaczne z:

```
1 def multiply(x, y):
2     return x*y
3
4 multiply(3, 5)
```

Przykładowe wywołanie i zwrócenie wartości z funkcji lambda, w inny sposób, bez przypisywania do zmiennej:

```
1 print((lambda x,y: (x*y)**2)(3,4))
```

Output:

```
1 144
```

Losowa tabelka, ponieważ nie było okazji do jej użycia

X	**2**	**3**	**4**
5	10	15	20
15	30	45	60
50	100	150	200

3.2 Funkcja map()

Funkcja map() pozwala na użycie danej funkcji dla każdego elementu iterowalnej struktury danych (np. listę lub krotkę).

Przykładowe użycia funkcji map():

```
1  def square(x):
2      return x*x
3
4  numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
5
6  answer = map(square, numbers)
7  print(answer)
8
9  numbersSquare = list(result)
10 print(numbersSquare)
```

Output:

```
1 <map object at 0x000002E3FFED8C10>
2 [1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Jak widać, przed wywołaniem wartości zwracanej przez map(), musimy przekonwertować ją na odpowiednią strukturę, np. list(), tuple() lub set().

Kwadrat liczb w liście za pomocą połączenia funkcji map() i Lambda:

```
1 numbers = (1, 2, 3, 4, 5, 6)
2 answer = map(lambda x: x**x, numbers)
3 print(list(answer))
```

Możemy także operować na wielu strukturach:

```
1 num1 = [1, 2, 3]
2 num2 = [5, 6, 7]
3 num3 = [10, 20, 30]
4
5 answer = map(lambda x, y, z: x + y + z, num1, num2, num3)
6 print(list(answer))
```

3.3 Funkcja filter()

Funkcja filter() zwraca elementy iteralnej struktury, dla których wartość wynosi True dla danej funkcji.

Przykładowe użycia funkcji:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]

def even(number):
    if number % 2 == 0:
          return True
    return False

print(tuple(filter(even, numbers)))
```

Output:

```
(2, 4, 6, 8)
```

Zwracanie liczb podzielnych przez 11 przy pomocy funkcji Lambda:

```
numbers = list(range(100))

answer = filter(lambda x: x % 11 == 0, numbers)

print(list(answer))
```

Output:

```
[0, 11, 22, 33, 44, 55, 66, 77, 88, 99]
```

Możemy też operować na literach:

```
letters = ['a', 'p', 'd', 'x', 'i', 'j', 'e', 'z', 's']

def check_word(letter):
    search = ['p', 'e', 'i', 'o', 'e', 's']
    return True if letter in search else False

word = filter(check_word, letters)

print(list(word))
```

Output:

```
['p', 'i', 'e', 's']
```

4 Inne przydatne rzeczy

4.1 f-strings

f-stringsy umożliwiają o wiele wygodniejsze i krótsze zapisywanie tekstu w funkcji printf()

```
>>> a = 10
>>> b = "<"
>>> f"a wynosi {a} i jest {b} od 100"
>>> a wynosi 10 i jest < od 100
```

4.2 Walrus operator

Walrus operator, opisywany za pomocą znaków ":="jest nową funkcjonalnością i został dodany w Pythonie 3.8. Pozwala na nadanie wartości zmiennej w obrębie danego wyrażenia.

```
a = 10
if (x := a) > 3:
    print(f"{x} jest > 3")
```

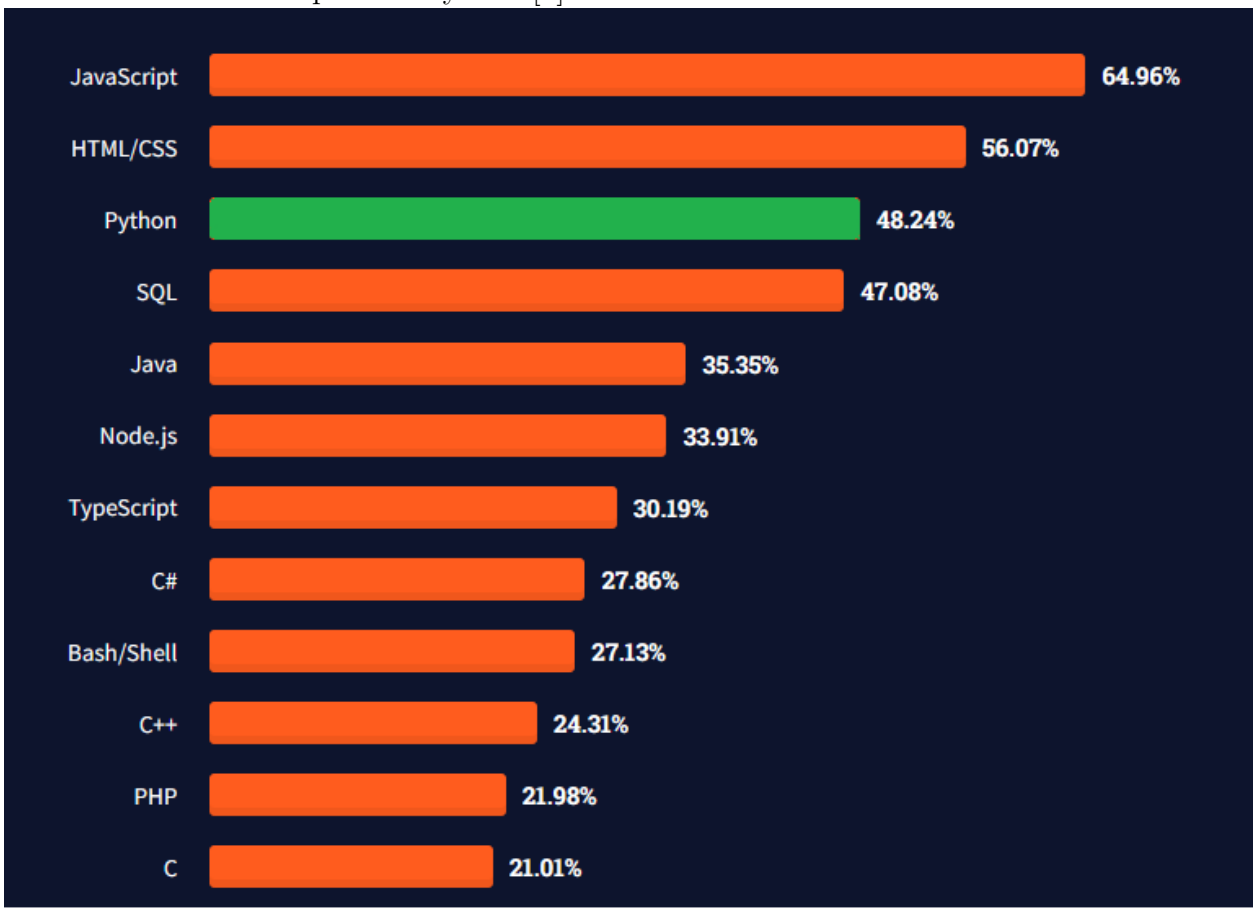
Operator może być użyteczny podczas pętli[3]:

```
sample_data = [
    {"userId": 1, "name": "rahul", "completed": False},
    {"userId": 1, "name": "rohit", "completed": False},
    {"userId": 1, "name": "ram", "completed": False},
    {"userId": 1, "name": "ravan", "completed": True}
]

print("With Python 3.8 Walrus Operator:")
for entry in sample_data:
    if name := entry.get("name"):
        print(f'Found name: "{name}"')

print("Without Walrus operator:")
for entry in sample_data:
    name = entry.get("name")
    if name:
        print(f'Found name: "{name}"')
```

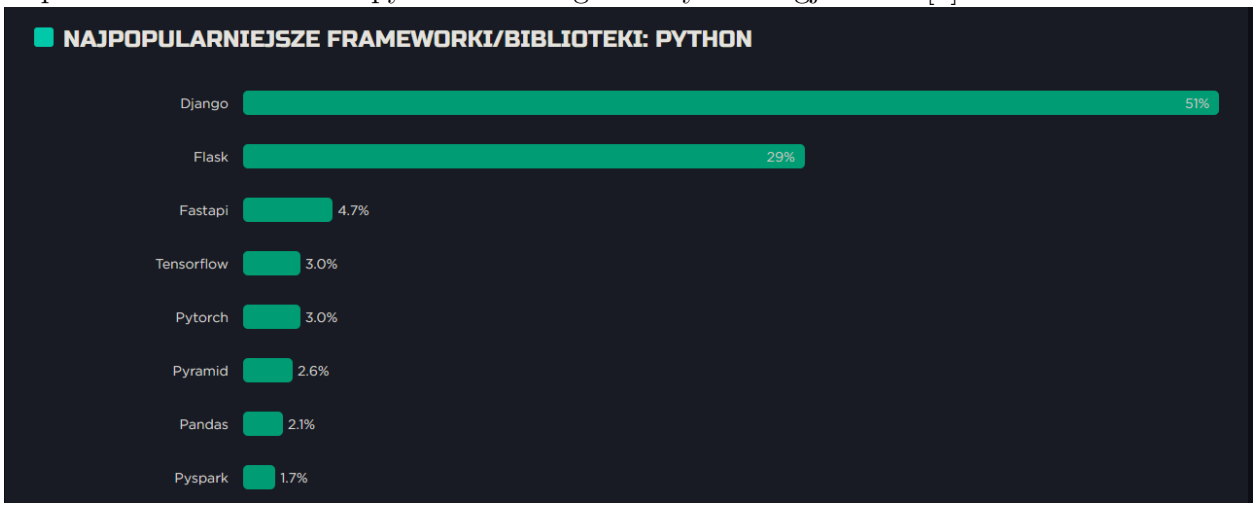
5 Popularność

Popularność pythona na tle innych języków według

Stack Overflow Developer Survey 2021[1]



Popularność frameworków pythona według ankiety bulldogjob 2021[2]



6 Bibliografia

[1] https://insights.stackoverflow.com/survey/2021most-popular-technologies-language

[2] https://bulldogjob.pl/it-report/2021/programmer

[3] https://www.geeksforgeeks.org/walrus-operator-in-python-3-8/

[4] https://en.wikipedia.org/wiki/Langtons$_a nt$