

CODE OMSCHRIJVING:

Willekeurige generator:

We gebruiken een zelfgeschreven generator voor willekeurige getallen om willekeurige getallen te genereren. Het gebruikt intern het Mersenne Twister-algoritme.

Beschrijving :

Initialisatie: Deze generator voor willekeurige getallen wordt geïnitieerd met een startwaarde, een geheel getal dat wordt gebruikt om het genereren van willekeurige getallen te starten. De seed-waarde wordt doorgaans door de gebruiker verstrekt of op een andere manier gegenereerd, zoals het gebruik van de systeemklok.

Staatsvector: Dit algoritme onderhoudt een toestandsvector, een array van 624 32-bits gehele getallen. De toestandsvector wordt geïnitieerd met behulp van de startwaarde en een reeks vaste parameters.

```
class Random_Number_Generator():
    def __init__(self, c_seed=0):
        (self.w, self.n, self.m, self.r) = (32, 624, 397, 31)
        self.a = 0x9908B0DF
        (self.u, self.d) = (11, 0xFFFFFFFF)
        (self.s, self.b) = (7, 0x9D2C5680)
        (self.t, self.c) = (15, 0xEFC60000)
        self.l = 18
        self.f = 1812433253
        # make a array to store the state of the generator
        self.MT = [0 for i in range(self.n)]
        self.index = self.n+1
        self.lower_mask = 0x7FFFFFFF
        self.upper_mask = 0x80000000
        # initial the seed
        self.c_seed = c_seed
        self.seed(c_seed)
```

Twist-functie: De belangrijkste functie van dit algoritme is de twist-functie, die de toestandsvector bijwerkt om een nieuwe reeks willekeurige getallen te genereren. De twist-functie werkt op een subset van de toestandsvector, waarnaar wordt verwezen als de "twist-matrix". De twistmatrix is een matrix van 32x624 die is afgeleid van de toestandsvector.

```
def twist(self):
    """ Generate the next n values from the series x_i """
    for i in range(0, self.n):
        x = (self.MT[i] & self.upper_mask) + \
            (self.MT[(i+1) % self.n] & self.lower_mask)
        xA = x >> 1
        if (x % 2) != 0:
            xA = xA ^ self.a
        self.MT[i] = self.MT[(i + self.m) % self.n] ^ xA
        self.index = 0
```

Uittreksfunctie: De extractiefunctie wordt gebruikt om een nieuw willekeurig getal uit de toestandsvector te genereren. De extractiefunctie werkt op het eerste element van de toestandsvector, een 32-bits

geheel getal. De extractiefunctie past een reeks bitsgewijze bewerkingen en wiskundige functies toe op het 32-bits gehele getal om een nieuw 32-bits geheel getal te genereren.

```
def extract_number(self):
    """ Extract a tempered value based on MT[index]
        calling twist() every n numbers
    """
    if self.index >= self.n:
        self.twist()

    y = self.MT[self.index]
    y = y ^ ((y >> self.u) & self.d)
    y = y ^ ((y << self.s) & self.b)
    y = y ^ ((y << self.t) & self.c)
    y = y ^ (y >> self.l)

    self.index += 1
    return y & 0xffffffff
```

Willekeurige functie: Deze functie geeft een willekeurig zwevend getal met een uniforme verdeling in het bereik van [0,1) (inclusief 0 en exclusief 1. Het bevat intern het 32-bits gehele getal van de bovenstaande functie

```
def random(self):
    """ return uniform ditribution in [0,1) """

    return self.extract_number() / 4294967296 # which is 2**w
```

Keuzes functie: De functie keuzes() retourneert een lijst met k elementen die willekeurig uit de populatie zijn gekozen. De elementen kunnen worden gekozen met of zonder vervanging, afhankelijk van de waarde van de gewichtparameter. Als er geen gewichten worden opgegeven, worden de elementen met gelijke waarschijnlijkheid gekozen.

```
def choices(self, population, weights=None, k=1):
    if weights is None:
        weights = [1] * len(population)
    elif len(weights) != len(population):
        raise ValueError("The number of weights must match the population size.")

    total = sum(weights)
    cumulative_weights = [0] + list(accumulate(weights, lambda x,y: x+y))

    choices = []
    for i in range(k):
        x = self.random() * total
        idx = bisect.bisect(cumulative_weights, x)
        choices.append(population[idx-1])

    if k == 1:
        return choices[0]
    return choices
```

Periodiciteit: Het Mersenne Twister-algoritme heeft een lange periode, het aantal willekeurige getallen dat kan worden gegenereerd voordat de reeks zich herhaalt. De periode van het Mersenne Twister-algoritme is $2^{19937}-1$, wat een zeer groot getal is.

SIMULATIE:

We hebben een dataset die bestaat uit tientallen landen genoemd als landenwoordenboek.

We hebben nog een hoofdwoordenboek met de naam as

Wedstrijden- Dit bevat het schema van het toernooi.

We zullen de klasse `random_number_generator` instantiëren met een seed-waarde (hier hebben we seed genomen als 243).

```
import pandas as pd

# Define the participating countries and their FIFA rankings
data = pd.read_excel("fifa-ranking.xlsx")
countries = dict(zip(data["country"], data["points"]))

countries['Iran'] = countries['IR Iran']
countries['South Korea'] = countries['Korea Republic']

RNG = Random_Number_Generator(243)

# Define the match schedule for the group stage

matches = {
    'Group A': [('Qatar', 'Ecuador'), ('Senegal', 'Netherlands'), ('Qatar', 'Senegal'), ('Netherlands', 'Ecuador'), ('Ecuador', 'Senegal'), ('Senegal', 'Qatar'), ('Netherlands', 'Qatar'), ('Qatar', 'Netherlands'), ('Ecuador', 'Netherlands'), ('Netherlands', 'Ecuador')],
    'Group B': [('England', 'Iran'), ('USA', 'Wales'), ('Wales', 'Iran'), ('England', 'USA'), ('Iran', 'USA'), ('Wales', 'England')],
    'Group C': [('Argentina', 'Saudi Arabia'), ('Mexico', 'Poland'), ('Poland', 'Saudi Arabia'), ('Argentina', 'Mexico'), ('Poland', 'Argentina'), ('Mexico', 'Poland'), ('Saudi Arabia', 'Poland'), ('Poland', 'Saudi Arabia'), ('Argentina', 'Mexico'), ('Mexico', 'Argentina'), ('Saudi Arabia', Argentina')],
    'Group D': [('Denmark', 'Tunisia'), ('France', 'Australia'), ('Tunisia', 'Australia'), ('France', 'Denmark'), ('Tunisia', 'France'), ('Denmark', 'Tunisia'), ('Australia', 'France'), ('France', 'Australia')],
    'Group E': [('Germany', 'Japan'), ('Spain', 'Costa Rica'), ('Japan', 'Costa Rica'), ('Spain', 'Germany'), ('Japan', 'Spain'), ('Costa Rica', 'Japan'), ('Germany', 'Spain'), ('Costa Rica', 'Germany')],
    'Group F': [('Morocco', 'Croatia'), ('Belgium', 'Canada'), ('Belgium', 'Morocco'), ('Croatia', 'Canada'), ('Croatia', 'Belgium'), ('Morocco', 'Croatia'), ('Canada', 'Belgium'), ('Belgium', 'Canada'), ('Morocco', Belgium')],
    'Group G': [('Switzerland', 'Cameroon'), ('Brazil', 'Serbia'), ('Cameroon', 'Serbia'), ('Brazil', 'Switzerland'), ('Serbia', 'Switzerland'), ('Switzerland', 'Brazil'), ('Cameroon', 'Brazil'), ('Serbia', 'Cameroon')],
    'Group H': [('Uruguay', 'South Korea'), ('Portugal', 'Ghana'), ('South Korea', 'Ghana'), ('Portugal', 'Uruguay'), ('South Korea', 'Portugal'), ('Ghana', 'Portugal'), ('Uruguay', 'South Korea'), ('Portugal', 'South Korea'), ('Ghana', 'South Korea')]
}
```

Er zijn drie functies

1) `simulate_match`

3) `conduct_matches`

2) `simulate_toernooi`

Simuleer_Match:

Wedstrijd simuleren is een functie die de winnaar van een wedstrijd simuleert met behulp van de poisson-verdeling.

We berekenen de lambda met behulp van de formule $(\text{teamA-punt}/\text{teamB-punt})$ die in de dataset wordt gegeven. Vervolgens simuleren we met behulp van de poissonverdeling de doelen voor elk team. Het team met de meeste doelpunten wordt als winnaar beschouwd en krijgt drie punten.

In het geval dat beide teams evenveel doelpunten maken, wordt één punt toegekend aan elk team.

```
# Define a function to simulate a match between two teams
def simulate_match(team1, team2):
    lambda_1 = countries[team1]/countries[team2]
    lambda_2 = countries[team2]/countries[team1]
    # Generate the number of goals scored by each team using a Poisson distribution
    goals1 = sum(RNG.choices(range(10), k=1000, weights=[math.exp(-lambda_1)*pow(lambda_1, i)/math.factorial(i) for i in range(10)]))
    goals2 = sum(RNG.choices(range(10), k=1000, weights=[math.exp(-lambda_2)*pow(lambda_2, i)/math.factorial(i) for i in range(10)]))

    # Determine the winner of the match
    if goals1 > goals2:
        return team1
    elif goals1 < goals2:
        return team2
    else:
        #simulate_match(team1, team2)
        return None
```

Wedstrijden uitvoeren:

Deze functie neemt het schema en geeft de winnaarsresultaten voor de respectievelijke groepen. We doorlopen het wedstrijdschema met de functie `Conduct_matches` en simuleren de winnaar voor elke wedstrijd met de functie `simulation_match`. En de scores voor elk land worden bijgewerkt in het `ind_results`-woordenboek. Nadat alle wedstrijden zijn voltooid, worden de 2 beste landen in elke groep gefilterd en opgeslagen in het `group_result`-woordenboek.

```
def conduct_matches(match_schedule):
    group_results = {}
    for group in match_schedule:
        ind_results = {}
        for match in match_schedule[group]:
            team1, team2 = match
            winner = simulate_match(team1, team2)
            if winner is not None:
                ind_results[winner] = ind_results.get(winner, 0) + 3
            else:
                ind_results[team1] = ind_results.get(team1, 0) + 1
                ind_results[team2] = ind_results.get(team2, 0) + 1

        # Sort the teams by their points in the group stage and simulate the knockout stage
        qualified_teams = sorted(ind_results.keys(), key=lambda x: ind_results[x], reverse=True)[:2]
        group_results[group] = qualified_teams

    return group_results
```

Simuleer_toernooi:

Eerst nemen we het schema van de groepsfase en sturen het naar de functie `conduct_matches`, daarna zal het de lijst met winnaars retourneren volgens de gegeven groepen die zijn opgeslagen in `group_stage_results`.

Daarna maken we het schema voor de knock-outfase en sturen het naar de functie `conduct_matches`, waarna het de lijst met winnaars retourneert volgens de gegeven groepen die zijn opgeslagen in het `knock_out_results`-woordenboek.

We doen vergelijkbare prijzen voor kwartalen, halve finales en finales. Aan het einde van de finale krijgen we de winnaar van het toernooi.

```

def simulate_tournament():
    # simulate the group stage
    group_stage_results = conduct_matches(matches)

    #simulate the knockout stage
    set_1 = [(i,j) for i,j in zip(group_stage_results['Group A'], group_stage_results['Group B'])]
    set_2 = [(i,j) for i,j in zip(group_stage_results['Group C'], group_stage_results['Group D'])]
    set_3 = [(i,j) for i,j in zip(group_stage_results['Group E'], group_stage_results['Group F'])]
    set_4 = [(i,j) for i,j in zip(group_stage_results['Group G'], group_stage_results['Group H'])]

    knock_out_schedule = {'set1':set_1,'set2':set_2,'set3':set_3,'set4':set_4}

    knock_out_results = conduct_matches(knock_out_schedule)

    #simulate quarters
    quarters_schedule = {'q1': [(i,j) for i,j in zip(knock_out_results['set1'],knock_out_results['set2'])],
                        'q2': [(i,j) for i,j in zip(knock_out_results['set3'],knock_out_results['set4'])]}

    quarter_results = conduct_matches(quarters_schedule)

    #simulate semi finals
    semi_schedule = {'s1':[quarter_results['q1']], 's2':[quarter_results['q2']]}

    semi_results = conduct_matches(semi_schedule)

    #conduct finals
    final_schedule = {'final':[(semi_results['s1'][0], semi_results['s2'][0])]}

    winner = conduct_matches(final_schedule)['final']
    if len(winner) == 1:
        return winner[0]
    else:
        return 'Draw'

```


We simuleren dit proces duizend keer en slaan de resultaten op in het resultatenwoordenboek. Na voltooiing van de simulatie printen we het aantal keren dat een land het toernooi wint. We printen ook het winnaarspercentage voor de landen.

```

: # Simulate the tournament 1000 times and count the number of wins for each team
results = {}
n_sim = 1000
for i in tqdm(range(n_sim),desc='Simulating', colour='blue'):
    winner = simulate_tournament()
    results[winner] = results.get(winner, 0) + 1

# Print the results
for team, count in results.items():
    print(f'{team}: {count} wins ({count/n_sim*100:.1f}%)')

```

Simulating: 100%|  | 1000/1000 [04:21<00:00, 3.82it/s]

Belgium: 580 wins (58.0%)
 France: 336 wins (33.6%)
 Brazil: 77 wins (7.7%)
 Draw: 7 wins (0.7%)