Rock Paper Scissors

Let's play the famous game against our computer.

https://en.wikipedia.org/wiki/Rock%E2%80%93paper%E2%80%93scissors (https://en.wikipedia.org/wiki/Rock%E2%80%93paper%E2%80%93scissors)

The use of functions is recommended

Goals

1. Use of loop
2. Data capture by console
3. Use if-elif-else
4. Use of try-except
5. Definition of functions. Modular programming
6. Logical operators.
7. Print
8. Import modules

In [ ]:

```
"""# Import the choice function of the random module
# https://stackoverflow.com/questions/306400/how-to-randomly-select-an-item-fr
om-a-list

import random

# Assign to a list the 3 possible options: 'stone', 'paper' or 'scissors'.

options = ['stone', 'paper', 'scissors']


# Assign a variable to the maximum number of games: 1, 3, 5, etc ...

max_games = 3


# Assign a variable to the number of games a player must win to win.
# Preferably the value will be based on the number of maximum games

must_win = 2


# Define a function that randomly returns one of the 3 options.
# This will correspond to the play of the machine. Totally random.

def play_game(lst):
    return random.choice(lst)

# Define a function that asks your choice: 'stone', 'paper' or 'scissors'
```

```
# you should only allow one of the 3 options. This is defensive programming.

# If it is not stone, paper or scissors keep asking until it is.

def ask_for_choice():
    choice = ""
    while choice == "":
        print ("Your turn:")
        a = input()
        if a in options:
            choice = a
    return choice


# Define a function that resolves a combat.
# Returns 0 if there is a tie, 1 if the machine wins, 2 if the human player wi
ns

def combat(options):
    machine_choice = play_game(options)
    human_choice = ask_for_choice()
    result = "tie"
    if machine_choice == human_choice:
        result = "tie"
    elif machine_choice == 'stone':
        if human_choice == 'paper':
            result = "human"
        else:
            result = "machine"
    elif machine_choice == 'paper':
        if human_choice == 'scissors':
            result = "human"
        else:
            result = "machine"
    else:
        if human_choice == 'stone':
            result = "human"
        else:
            result = "machine"
    return [result, machine_choice, human_choice]


# Define a function that shows the choice of each player and the state of the
game
# This function should be used every time accumulated points are updated
def play_complete_game(options):
    number_games = 0
    results_list = []
    wins_human = 0
    wins_machine = 0
    while (wins_human < 2) and (wins_machine < 2):
        number_games += 1
        start_combat = combat(options)
        results_list.append(start_combat[0])
        print ("Game nº {number_games}: M:{choice_machine} - H:{choice_human}.
Accumulated results: {acumulated_results}".format(number_games = number_games,
choice_machine = start_combat[1], choice_human = start_combat[2], acumulated_r
esults = results_list))
```

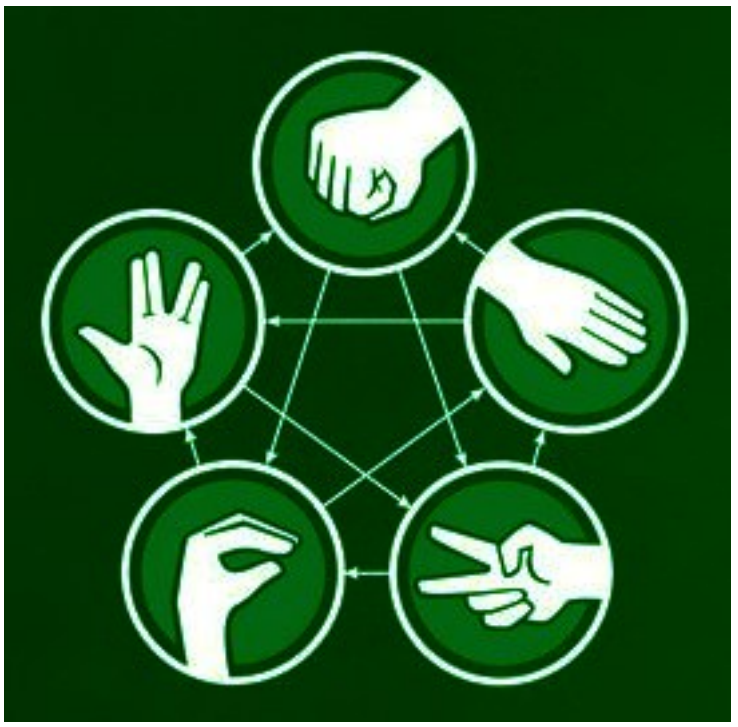
```
        if start_combat[0] == "machine":
            wins_machine += 1
        elif start_combat[0] == "human":
            wins_human += 1
    if wins_human > wins_machine:
        winner = "human"
    elif wins_human == wins_machine:
        winner = "no one"
    else:
        winner = "machine"
    return "The game is finished, {winner} wins!".format(winner = winner)

print(play_complete_game(options))

# Create two variables that accumulate the wins of each participant


# Create a loop that iterates while no player reaches the minimum of wins
# necessary to win. Inside the loop solves the play of the
# machine and ask the player's. Compare them and update the value of the varia
bles
# that accumulate the wins of each participant.


# Print by console the winner of the game based on who has more accumulated wi
ns
"""
```

```
Your turn:
```

Expected output: Depends on the inputs, you know how to play and what to expect.

Bonus: Stone, paper, scissors, lizard, spock

Now the improvement begins.



http://www.samkass.com/theories/RPSSL.html (http://www.samkass.com/theories/RPSSL.html)

You are asked to impliment some improvements with respect to the simple previous game. In addition, the number of games (which must be ODD) will be requested per console until a valid number is entered.

Improvements:

- 5 options: stone, paper, scissors, lizard, spock
- The number of games is requested per console Tip: Reuse code that you already use. If you have programmed intelligently, the bonus are simple modifications to the original game.

In [2]:

```
# Import the choice function of the random module


# Define a function that asks for an odd number on the keyboard, until it is n
ot valid
# will keep asking


# Assign a list of 5 possible options.


# Assign a variable to the maximum number of games: 1, 3, 5, etc ...
# This time the previously defined function is used


# Assign a variable to the number of games a player must win to win.
# Preferably the value will be based on the number of maximum games


# Define a function that randomly returns one of the 5 options.
# This will correspond to the play of the machine. Totally random.
```

```
# Define a function that asks your choice between 5
# you should only allow one of the 5 options. This is defensive programming.
# If it is not valid, keep asking until it is valid.


# Define a function that resolves a combat.
# Returns 0 if there is a tie, 1 if the machine wins, 2 if the human player wi
ns
# Now there are more options



# Define a function that shows the choice of each player and the state of the
game
# This function should be used every time accumulated points are updated


# Create two variables that accumulate the wins of each participant


# Create a loop that iterates while no player reaches the minimum of wins
# necessary to win. Inside the loop solves the play of the
# machine and ask the player's. Compare them and update the value of the varia
bles
# that accumulate the wins of each participant.


# Print by console the winner of the game based on who has more accumulated wi
ns
```

```
import random

options = ['stone', 'paper', 'scissors', 'lizard', 'spock']

max_games = 3

must_win = 2

def play_game(lst):
    return random.choice(lst)


def ask_for_choice():
    choice = ""
    while choice == "":
        print ("Your turn:")
        a = input()
        if a in options:
            choice = a
    return choice

def combat(options):
    machine_choice = play_game(options)
```

```
human_choice = ask_for_choice()

result = "tie"
if machine_choice == human_choice:
    result = "tie"
elif machine_choice == 'stone':
    if human_choice == 'paper' or human_choice == 'spock':
        result = "human"
    else:
        result = "machine"
elif machine_choice == 'paper':
    if human_choice == 'scissors' or human_choice == 'lizard':
        result = "human"
    else:
        result = "machine"
elif machine_choice == 'scissors':
    if human_choice == 'stone' or human_choice == 'spock':
        result = "human"
    else:
        result = "machine"
elif machine_choice == 'lizard':
    if human_choice == 'stone' or human_choice == "scissors":
        result = "human"
    else:
        result = "machine"
else:
    if human_choice == "lizard" or human_choice == "paper":
        result = "human"
    else:
        result = "machine"
    return [result, machine_choice, human_choice]


def ask_max_number_games():
    max_number_games = 0
    while max_number_games == 0:
        print ("How many games do you want to play:")
        a = input()
        try:
            max_number_games += int(a)
        except ValueError:
            pass
    return max_number_games

def play_complete_game(options):
    max_number_games_game = ask_max_number_games()
    number_games = 0
    results_list = []
    wins_human = 0
    wins_machine = 0
    while number_games < max_number_games_game:
        number_games += 1
        start_combat = combat(options)
        results_list.append(start_combat[0])
        print ("Game nº {number_games}: M:{choice_machine} - H:{choice_human}.
Accumulated results: {acumulated_results}".format(number_games = number_games,
choice_machine = start_combat[1], choice_human = start_combat[2], acumulated_r
esults = results_list))
```

```
        if start_combat[0] == "machine":
            wins_machine += 1
        elif start_combat[0] == "human":
            wins_human += 1
    if wins_human > wins_machine:
        winner = "human"
    elif wins_human == wins_machine:
        winner = "no one"
    else:
        winner = "machine"
    return "The game is finished, {winner} wins!".format(winner = winner)

print(play_complete_game(options))
```

```
How many games do you want to play:
fd
How many games do you want to play:
4
Your turn:
lizard
Game nº 1: M:spock - H:lizard. Accumulated results: ['human']
Your turn:
spock
Game nº 2: M:lizard - H:spock. Accumulated results: ['human', 'mac
hine']
Your turn:
rock
Your turn:
stone
Game nº 3: M:scissors - H:stone. Accumulated results: ['human', 'm
achine', 'human']
Your turn:
paper
Game nº 4: M:stone - H:paper. Accumulated results: ['human', 'mach
ine', 'human', 'human']
The game is finished, human wins!
```

Expected output: Depends on the inputs, you know how to play and what to expect.