Girls' Programming Network

Scissors Paper Rock!

This project was created by GPN Australia for GPN sites all around Australia!

This workbook and related materials were created by tutors at:

Sydney, Canberra and Perth

Girls' Programming Network

If you see any of the following tutors don't forget to thank them!!

Writers

Sarah Mac
Renee Noble
Vivian Dang
Courtney Ross
Olivia Ross
Kimberley Apted

Testers

A massive thanks to our sponsors for supporting us!

ATLASSIAN

Part 0: Setting up

Task 0.1: Making a python file

Open the start menu, and type ‘IDLE’. Select IDLE 3.6.

1. Go to the File menu.
2. Select ‘New Window’ to open a new window.
3. Go to the File menu in the new window.
4. Select ‘Save As...’ and save the file as ‘SPR.py’.

Task 0.2: You’ve got a blank space, so write your name!

At the top of the file use a comment to write your name!
Any line starting with # is a comment.

```
# This is a comment
```

CHECKPOINT ☑

If you can tick all of these off you can go to Part 1:

- ☐ You should have a file called SPR.py
- ☐ Your file has your name at the top in a comment
- ☐ Run your file with F5 key and it does nothing!!

Part 1: Welcome Message

Task 1.1: Print a welcome and the rules

Welcome the player and print the rules!

Use a print to make it happen when you run your code:

```
------------------------------------------------------
Welcome to Human vs. Computer in Scissors, Paper, Rock!
------------------------------------------------------
Moves: choose scissors, paper or rock by typing in your selection.
Rules: scissors cuts paper, paper covers rock and rock crushes scissors.
Good luck!
------------
```

Don't want to type all that out? Go to this link: http://bit.ly/2nzHvVM

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 2:

- ☐ Print a welcome
- ☐ Print the rules
- ☐ Try running your code!

2. Who played what?

Classes

Task 2.1: Players can be humans or computers

Create a `Player` class that will store the name and move of each player. It should include an `__init__` function that takes the player's name as an argument. We don't know what the player's move is yet, so we can set it to `None`.

Create two `Player` instances, one in a variable called `human` and another in a variable called `computer`. For now just pass in the names `"Human"` and `"Computer"`.

Hint

Remember to use the `self` keyword as the first argument for the functions belonging to the class (including `__init__`) and for defining any variables belonging to the class (such as `self.name`).

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

Here is an example of how to create an instance of the `Person` class. It is calling the special `__init__` function, known as a *constructor*.

```
kim = Person("Kim", 33)
```

Remember:

- The `kim` variable is an instance of the object class `Person`
- You can have as many instances of the `Person` class as you like, each with their own name and age. For example, you could make one for every girl at GPN!

Task 2.2: Make the computer play the same move every time!!

Add a `set_move` function to the `Player` class, and use this function to set the move for the `computer` player to "scissors", "paper" or "rock".

Hint

Here is an example of how to call a function called `add` function that has been defined for a class called `Counter`. Notice that the `self` keyword is *not* used outside the class.

```
c = Counter()
c.add(20)
```

Task 2.3: Ask the human for their move

Use `input` to ask the human for their move and save their answer in the human player instance.

It should now look like this when you run your code:

```
------------------------------------------------------
Welcome to Human vs. Computer in Scissors, Paper, Rock!
------------------------------------------------------
Moves: choose scissors, paper or rock by typing in your selection.
Rules: scissors cuts paper, paper covers rock and rock crushes scissors.
Good luck!
------------

What is your move? scissors, paper or rock? rock
```

Task 2.4: Print out the moves

Print out the moves the computer and the human have played. Remember to use the name we have stored in the player instances.

It should now look like this when you run your code:

```
------------------------------------------------------
Welcome to Human vs. Computer in Scissors, Paper, Rock!
------------------------------------------------------
Moves: choose scissors, paper or rock by typing in your selection.
Rules: scissors cuts paper, paper covers rock and rock crushes scissors.
Good luck!
------------

What is your move? scissors, paper or rock? rock

Computer played: scissors
Human played: rock
```

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 3:

- ☐ Your program contains a Player class
- ☐ There is a variable for the human player and the computer player
- ☐ Set a move for the computer
- ☐ Ask the human to type in their move and set their response
- ☐ Print out the human and computers moves
- ☐ Run your code!

★ BONUS 2.5: Personalise the game

Waiting for the next lecture? Try adding this bonus feature!!

At the start of the game ask the human to enter their name, and use this as the argument to create the human player.

It should now look like this when you run your code:

```
-----------------------------------------------------
Welcome to Human vs. Computer in Scissors, Paper, Rock!
-----------------------------------------------------
Moves: choose scissors, paper or rock by typing in your selection.
Rules: scissors cuts paper, paper covers rock and rock crushes scissors.
Good luck!
------------

What is your name? Kim

What is your move? scissors, paper or rock? rock

Computer played: scissors
Kim played: rock
```

3. Win, lose or tie?

Let's figure out who won the game!

Task 3.1: What are the different ways to win, lose and tie?

What are all the combinations of how the game could go? Finish this table:

Human Move	Computer Move	Who Wins?
scissors	scissors	draw
scissors	paper	human
scissors	rock	
paper		

Task 3.2: Add the players to a game

Create a `Game` class with an `__init__` function that will take two players as arguments and store them.

Dictionaries

Task 3.3: Store the winning moves in a dictionary

Add a static variable to `Game` called `win_moves` which should contain a dictionary of all the winning moves.

To find the winning moves, identify the rows where the human wins. You can then use the human moves for the keys and the corresponding computer moves for the value.

Hint

To add a static variable you *don't* need to use the `self` keyword, and you don't need to create an instance in order to access static variables. Here is an example of a class with a static dictionary:

```
class MyClassName:
    my_dict = {'key1': 'value1', 'key2': 'value2'}

print(MyClassName.my_dict['key1'])
```

Task 3.4: Get the winner of the match

Add a `get_match_winner` function to the `Game` class that returns `None` if it's a tie, or else the player object with the winning move.

Use `if`, `elif` and `else` statements to choose the correct return value:

- **If** both moves are the same, then the game is a tie.
- **Else, if** the first players move is the key to the second players move in the `win_moves` dictionary, then the first player is the winner.
- **Else** the second player is the winner!

Hint

Use the `return` keyword in your function to send a value back to the code that called the function. Here is an example of a function that returns the word "Fizz" if the argument provided is divisible by 3:

```
def fizz(x):
    if x % 3 == 0:
        return "Fizz"
    else:
        return x

result = fizz(13)    # Store the returned value
print(fizz(9))       # Or do something with the returned value
```

Task 3.5: Announce the winner!

Store the result from `get_match_winner` in a variable, and use this to print out the winner of the match.

- If there is no winner print "It's a tie!"
- Otherwise print the winner's name followed by "won the match"

Make sure you do this *after* you have set the moves for both the human and computer players.

Hint

You will need to create an instance of `Game` with your human and computer players before you can call the `get_match_winner` function.

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 4:

- ☐ Your program contains a Game class
- ☐ Game has a dictionary containing winning move combinations
- ☐ Game has a function that selects a winner between two players
- ☐ Create a dictionary containing every combination of moves
- ☐ Store who won in a variable
- ☐ Print out the winner
- ☐ Run your code and test different moves!
- ☐ Test when you input “ROCK” or “Rock” instead of “rock”, what happens?

★ BONUS 3.6: ROCK Rock rOcK!

Waiting for the next lecture? Try adding this bonus feature!!

We see that **“Rock”** is not the same as **“rock”** and our game only works when it’s all in lowercase.

Try to make your game work when player input a move with capital letters such as **“Rock”** or **“sCissors”**. Think about how you’d convert them to all lowercase.

★ CHALLENGE 3.7: Who has time for all this typing!?

It would be far more convenient for the user if they didn’t have to type in the whole word for their move.

- Let’s *transform* the input to be *standardised*.
- What if the user only puts in one or two letters? “S” “ro” “PP”

4. Smarter Computer

The computer keeps playing the same move! That's no fun! Let's make the computer chose a random move!

Task 4.1: Import random library

To get access to cool random things we need to import random!

At the top of your file add this line:

```
import random
```

Task 4.2: Chose a random move!

Find your line of code where you set your computer move, improve this line by choosing a random move.

Use chose a random move for the computer using `random.choice` from a list of "paper", "scissors" and "rock".

Hint

If I wanted to choose a random food for dinner I could use code like this:

```
dinner = random.choice(["pizza", "chocolate", "nutella",
"lemon"])
```

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 5:

- ☐ The computer plays a random move every time.
- ☐ The line "Computer played: …." prints different things out!
- ☐ Try different moves against the computer, does the the correct winner print?

5. Again, Again, Again!

We want to play Scissors-Paper-Rock more than once! Let's add a loop to play on repeat!

Task 5.1: How many games?

Find out how many games the user wants to play. Put this after your welcome message!

Hint

Input returns a **string**. Make sure you **convert it to an int** and store it in a variable!

Remember `int("57")` will give you back 57. You can use `int(...)` on a variable too!

For Loops

Task 5.2: Loop time!

Create a for loop that runs as many times as the user asked for.

Hint

You will need to use a `for` loop and the `range(...)` function to specify the number of times to repeat the steps in your code. Here is an example:

```
for i in range(10):
    print(i)          # This will be repeated 10 times
print("All done!")    # This will not be repeated
```

Remember:

- You can use a variable with `range(...)` instead of 10.
- Things we want to do every game must be *indented* to be included in the loop.

Task 5.3: GAME OVER!

After all the rounds are played, print out "GAME OVER!".
Make sure this is after your loop and doesn't print every round!

☑ CHECKPOINT ☑

If you can tick all of these off you can go to the Part 6:

- ☐ Ask the user how many games they want to play
- ☐ Your game repeats the number of times the user asked for
- ☐ GAME OVER prints once, after all of the rounds!
- ☐ Test when the user inputs something “one” instead of 1 for the number of games they want to play, what happens?

★ CHALLENGE 5.4: “One” is the loneliest number

Waiting for the next lecture? Try adding this bonus feature!!

We found that our game will **crash** if the user does not input a valid number when choosing the number of games to play.

```
How many matches should win the game? one
Traceback (most recent call last):
  File "spr_adv_game_class.py", line 84, in <module>
    target_score = int(input("How many matches should win the game? "))
ValueError: invalid literal for int() with base 10: 'one'
```

We can make our program more user friendly by providing instructions for what happens when the `int(...)` function results in a `ValueError`, and we can do this using the `try` and `except` statements.

- The `try` statement defines a block of code where we want to watch out for particular errors.
- The `except` statement provides instructions for the program to follow if the specified error is encountered in the `try` block.

Here is an example from the Python online documentation that will run a loop until the user enters a valid number:

```
while True:
    try:
        x = int(input("Please enter a number: "))
        break
    except ValueError:
        print("Oops!  That was not a valid number.  Try again...")
```

Try applying this example to your code!

6. Keeping Score!

Why play lots of games if we're not even keeping count of who wins?? Let's keep score!

Task 6.1: Counter!

Add a `score` variable to the `Player` class. Make sure you set it to 0 in the `__init__` function!

Task 6.2: Add 1!

Add an `add_win` function to the `Player` class that will increase `score` by 1 each time it is called.

Every time the computer or human wins we need to call `add_win` on the correct instance for the winning player. If it's a tie neither player gets a point!

Hint

You added some code in Task 3.5 to print out whether the match was a tie or to announce the winner. This would be a great place to update the score too!

Task 6.3: Get the winner

Add a `get_game_winner` function to the `Game` class that compares the scores of the two players and returns the player with the highest score.

Hint

Refer back to Task 3.4 for hints on returning values from functions.

Task 6.4: And the winner is...

After all the games are played we need to report the over all winner.

- Print out how many games the human and computer won each.
- Call the `get_game_winner` function and print out who the overall winner was!
- Print a neutral message if both players tied.

The game over message should now looks something like this:

```
------------------------------------------------------
GAME OVER!
Kim won 5 matches
Computer won 2 matches
Kim is the winner!!
------------------------------------------------------
```

☑ CHECKPOINT ☑

If you can tick all of these off you can go to the Part 6:

- ☐ Player has a score variable which starts at 0
- ☐ Player has a function which increases the players score by 1
- ☐ Game has a function that selects the winner with the highest score
- ☐ Print the final score for each player at the end of the game
- ☐ Print the overall winner at the end of the game

★ CHALLENGE 6.5: First to X

Right now we play a set number of games. But can you figure out how you could change your program to keep playing until a player gets to a certain number of points?

You might need to use a `while` loop, or a `break`, or something else you can think of!

7. That's not a real move!

What happens if the human plays a wrong move, like Batman? Or does a typo, like "ppaer"? Test your code and find out!

We need to make our code more robust! If you haven't already, also make sure you also go back and do Task 3.6.

While Loops

Task 7.1: We're repeating ourselves - let's make a function

Asking the player for input shouldn't be hanging out the main game. It belongs in a function!

1) Move your block of code asking what the user wants to do into a function in the `Player` class.
2) Start the function with an input prompt
3) Then: `while` the variable is `not in` the list of acceptable answers we're going to ask again.
4) Once we get an acceptable answer, store it in `self.move`

☑ CHECKPOINT ☑

If you can tick all of these off you can go to the extensions:

- ☐ Input is transformed to upper or lower case
- ☐ Input is checked against list of allowed values
- ☐ Loop if the input is not in the list

★ CHALLENGE 7.2: Game Over! Shut Down!

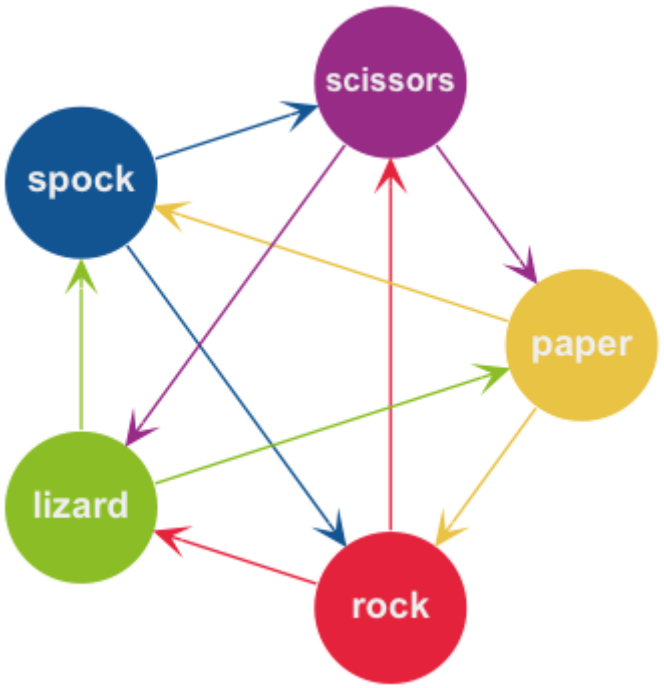
Sometime the user might say they want to play a certain number of rounds, but has to leave before the rounds are finished.

Create an `if` statement that checks to see if the user entered "quit" as their move, and close the game down.

Don't forget to tell the user who the overall winner was!

8. Extension: Scissors, Paper, Rock, Lizard, Spock!

Let's add some more moves and play Scissors, Paper, Rock, Lizard, Spock! Follow the arrows in the picture to see who wins!



Task 8.1 Updated moves!

When you ask the user what move they want to play, include lizard and spock!

Make sure you give the computer the same options!

Task 8.2 Updated combos!

Update the moves dictionary to include all of the new combinations!

9. Extension: AI. The computer reads your mind!

Let's make the game more challenging by having the computer guess what move the user will choose next, based on what the user has chosen before!

Task 9.1 Create the dictionary

Create an empty dictionary called `ai` for the computer. Store the move the human last chose in another variable, such as `last_move`.

Task 9.2 Store what happens next!

In the dictionary, add the move that the human played last round as a `key` if it's not already in the dictionary. Then store the move that the human played this round in a `list` as the `value`.

Make sure this happens every round!

Task 9.3 Get the computer to choose!

Now that the computer knows what the human played last time, get it to guess what you'll play next!

If the move the human played last is in the `ai` dictionary, choose the computer's move from the list of values. Otherwise, select the `computer_move` from the standard lists of moves!

★ CHALLENGE 9.4: Pick the winner

Our computer move will now pick the more likely move to make it tie with the human. But we want our computer to win!

Rather than storing what the human played as the value in the dictionary, store the move that would win against the human instead.