Canberra Girls' Programming Network

Scissors-Paper-Rock Workbook

Have you ever wondered how to write a computer game? Well today you'll write the code for your very own game! Today we'll write a “text-based” game, which means the game is all text.

We'll be using Python to write our game! Specifically we’ll be using Python 3 and an editor called IDLE. Both of these are free, which means you can install them on any computer you want! If you want to learn more about Python, or write more games, you can install Python at home!

You can take this workbook home with you, and if you don’t complete all of the activities today, you can complete them at home.

0. Setting up

Before we get started we need a Python file. Python programs are written in files with the .py file extension. Go to the following website and download the Python file template:

https://cbrgpn.wordpress.com/

Save it to the desktop and rename it to firstname_lastname_spr.py.

Next we need to open our editor. We're going to be working in an editor called "IDLE". The editor highlights code in different colours which makes reading and writing Python code a lot easier!

1. Open the launchpad by clicking on the rocket on the bottom left of the screen.
2. Search for “terminal” and run the terminal.
3. Type “idle3” and press Enter to launch IDLE.
4. IDLE will open. The window you see initially is called the Python shell. More on that later.
5. Open the .py file you just downloaded.

Let’s run it! Click Run > Run module, or fn F5. You should see this printed in the Python shell:

```
Welcome to Python!
```

Congratulations, you just ran your first Python program!

Now we can get started coding!

1. How does the game work?

On each turn, the two players choose one of three moves: Scissors, Paper or Rock. Each move has the ability to defeat another move according to the following rules:

- Scissors cuts paper.
- Paper covers rock.
- Rock destroys scissors.

If both players choose the same move (for example, paper against paper), then it's a tie.

We're going to write a program where we can play Scissors-Paper-Rock against the computer. On each turn, we'll be able to choose a move, and the computer will choose one too. Then our program will tell us whether we won, lost, or tied on that turn.

Have a look at the code you have open in IDLE. So far, it consists of a lot of lines starting with # (don't worry about these just yet), and one other line:

```
print("Welcome to Python!")
```

This line tells the program to print a message to the user. A line of code that looks like this, with a name (in this case the name of the function is "print") and a set of parentheses (), is called a **function**. A function uses the information inside the parentheses and performs something (hopefully!) useful. In this case, it prints something to the Python shell. We will encounter many functions like this that we can use for free in Python, but the `print()` function is one we will use a lot.

Tip! What is this thing we keep referring to called the "Python shell"? The Python shell is an interactive window where you can test out small bits of Python code and run them immediately. Try it. Type `print("Welcome to the Python shell!")` and hit Enter and see what happens.

Now, instead of this message, we would like to tell the human player the rules of Scissors-Paper-Rock when they start the game.

Exercise 1: Print a welcome message

Edit the `print()` statement in the editor so that the game prints the following message to the user:

```
-----------------------------------------------------
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!
------------
```

You may need to use several `print()` statements since it is several lines long! Run your program by clicking Run > Run module, or fn F5.

When you run your program at this point it should:

- ✓ Print out the game title, rules and allowed moves.

Congratulations, you just *wrote* your first Python program!

2. Who played what?

Now we know the rules, we want to start to store the human's move, and the computer's move. To do this we will need **variables**. A variable is basically a name that is assigned a value. A variable can be assigned to a number, some text, or something more complex like a list. Variables should always have names that make sense when you, or someone else, reads your code!

Here are some examples of creating variables:

```
some_number = 5
some_text = "a bit of text"
```

In this example, `some_number` and `some_text` are the variable names.

That's it, pretty simple. Notice that text is always surrounded by quotation marks, but numbers aren't. Try writing these statements in the Python shell and see what happens. After this, we can use `some_number` and `some_text` in other code. Try writing `print("The number is", some_number)` and `print("The text is", some_text)` in the shell and see what happens.

We need to make two variables: one to store the human's move and another to store the computer's move. This lets us keep track of who played what, and compare them later to determine who wins each round.

Exercise 2: Store the moves

1. Make a variable for the human's move, and store one of the possible moves in it, e.g. "scissors", "paper" or "rock".
2. Do the same for the computer's move.
3. Print out each move. It should look something like this, depending on what you stored in each variable:

```
human plays: scissors
computer plays: rock
```

When you run your program at this point it should:

- ✓ Print out the game title, rules and allowed moves.
- ✓ Print out the human's and computer's moves.

Tip! What are those lines starting with `#`? These are called comments. They are ignored by the computer so you can write anything you like on them. You can (and should!) add comments to your program to describe to everyone what your program is supposed to do.

At the moment we have to change our code every time we want to try different moves. Instead, let's use the `input()` function to ask the human what move they want to play. The `input()` function is another inbuilt function, like `print()`. It prompts the user to supply information to the computer program. For example if you wanted a user to input their age, you would write:

```
user_age = input("What is your age? ")
print("Your age is",user_age)
```

This line would print `What is your age?`, then assign whatever the user typed to the variable `user_age`. The next line will print `Your age is` followed by whatever the user typed in.

Exercise 3: Ask the human for their move

1. Ask the human for their move using the `input()` function, and assign it to the variable you created in exercise 2 for the human's move.

When you run your program at this point it should:

- ✓ Print out the game title, rules and allowed moves.
- ✓ Ask the user what move they want to play.
- ✓ Print out the human's and computer's moves.

3. Win, lose or tie?

One of the most important parts of computer programs is making decisions about what to do next. These decisions might depend on what a user does, what is in a file or database, or something else. In Scissors-Paper-Rock, we need to decide who won. To do this, we need **if** statements. An if statement checks if something is true, and if so it does something, if not, then it does something else. Here's an example:

```
1  year_at_school = 11
2  if year_at_school == 11:
3      print("You're in year 11.")
4  else:
5      print("You're not in year 11.")
```

There are a few things going on here, so let's break it down. First, we made a variable called `year_at_school` and assigned it the number 11 in line 1. That's nothing new. Then, we start the if statement on line 2.

If statements describe conditions that are either met or not met. In line 2 the condition we are checking is if the variable `year_at_school` is equal to 11.

The `==` is the equality operator. Don't confuse this with `=`, which is the assignment operator! The equality operator compares if two things are equal.

If the condition on Line 2 is met, (which it has been in this example), then we run the code that is on line 3. If the condition has not been met, that is they are not equal, then the block of code starting on line 5 is run.

In Python, the if condition always ends with the colon symbol. The indentation on lines 3 and 5 is important as it shows where the next block of code starts, if the condition on the line before has been met.

What happens when there is more than one condition to be met?

```
1  year_at_school = 11
2  if year_at_school == 11:
3      print("You are in college in year 11.")
4  elif year_at_school == 12:
5      print("You are in college in year 12.")
6  else:
7      print("You are not in college.")
```

In line 4, we add an `elif` block. This is shorthand for "else if" and basically means, if the first thing wasn't met, then check if this other thing is met. You can have as many `elif` statements or conditions as you want.

Sometimes there are multiple conditions to be met to satisfy the if statement.

```
1  year_at_school = 12
2  studying_maths = True
3  if year_at_school == 11 and studying_maths:
4      print("You are studying year 11 maths.")
5  elif year_at_school == 12 and studying_maths:
6      print("You are studying year 12 maths.")
7  else:
8      print("You're either not studying maths or not at college.")
```

In line 3, we are combining two comparisons using the `and` operator. If both of the conditions are met, then the if statement has been satisfied and the next line of code (that is indented) is run. Which line do you think should print in the last example? Try it in the shell and see if you're right. (It can be tricky adding multi-line commands to the shell, try just adding it one line at a time.)

We can also combine the comparisons using the `or` operator. When there is an `or` operator, if either one of the conditions are met, then the if statement is satisfied and the indented code is run.

OK, we are nearly ready to work out who has won! In Scissors-Paper-Rock there are three ways for the human to win, three ways for the human to lose, and three ways to draw. You'll need to

check for all of these cases in your if statements! If you're not sure, try writing them out on paper first, like this:

If human plays scissors, and computer plays scissors, it's a tie.
Else, if human plays scissors, and computer plays paper, human wins.
Else, if human plays scissors, and computer plays rock, human loses.
...

You can fill in the rest. Writing things out like this in plain language is called writing pseudo code. It's a handy way to design programs without needing to worry about getting the syntax right.

Exercise 4: Determine the winner

Compare the human's move to the computer's move using `if` statements and the `and` operator:

- If the human player wins, then print a message saying "You won!"
- If the computer wins, then print a message saying "You lost."
- If it is a tie, then print a message saying "You tied with the computer."

Tip! You can use the print statement to test your code at any stage of your work. You can even print out the result of a comparison, like `print(human_move == "scissors" and computer_move == "rock")`. It will either print `True` or `False`. Then you can comment them out by using `#` at the start of the line, or simply delete the block of test code. We call these print statements "debug output" or "print debug statements".

When you run your program at this point it should:

- ✓ Print out the game title, rules and allowed moves.
- ✓ Ask the user what move they want to play.
- ✓ Print out the human's and computer's moves.
- ✓ If the human wins, print out "You won!"

- ✓ If the computer wins, print out “You lost.”
- ✓ If it is a tie, print out “You tied with the computer.”

One of the most important things about writing programs is testing them. There is no use writing some code if you don’t know that it does what you think it should do. Testing is how we discover bugs and things that we might have overlooked when designing the program.

Exercise 5: Test your program

Try inputting a few different things for the human move, and see what happens. Does it do what you expect, or what you think it should do? If it doesn’t, why not? Try the following:

- “scissors”, “paper” or “rock”
- “Scissors”, “SCISSORS” or “sCissors”
- “banana” or “batman”
- nothing (Enter)

Try and fix anything you think you can fix, and just make a note of the rest. There’s an advanced exercise later that will make the code more robust to these sort of things. From here on, you should be testing your code after every exercise to check that it does what you think it should do.

4. Smarter computer

Playing with a computer is not very challenging if the computer plays the same move all the time. It would be far more fun if the computer could chose a move randomly! We're going to change the code so that the computer chooses a move randomly from the set of allowed moves. To do this, we are going to need **lists** and to **import** a module.

So far we have dealt with variables that are either numbers, or text. Another type of variable is a list. A list is as it sounds: a list of things. These things can be numbers, text, or even lists! Yes, you can have a list of lists. You create a list with square brackets and separate each item with a comma, like this:

```
dinner_options = ["spaghetti", "tacos", "stir-fry"]
```

There are a lot of functions that you can use to change or access a list's items. We don't need any of them just yet, but you might need to know more about lists to complete the advanced exercises.

Tip! To find out more details about anything mentioned in this workbook, you can always refer to the Python documentation at https://docs.python.org/3/ (remember, we are using Python 3, not Python 2). You will probably find the Tutorial, Library Reference and Language Reference most useful. For example, to learn more about what you can do with a list, check out https://docs.python.org/3/tutorial/introduction.html#lists or https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range.

Now, let's say you couldn't decide what to have to dinner and you wanted to choose randomly. For this, we can call on the help of some other code handily written by somebody else. We do this by importing a module. The module that we need is called random, so we add this at the very top of our code:

```
import random
```

Now, to choose a random item from a list, we use the `random.choice()` function, like this:

```
dinner = random.choice(dinner_options)
```

So, our full dinner selection example looks like this:

```
import random
dinner_options = ["spaghetti", "tacos", "stir-fry"]
dinner = random.choice(dinner_options)
print(dinner)
```

Try it in the shell and see what Python has chosen for your dinner! Try it a few times, and see if it picks something different each time. We are now ready to make our computer move random!

Exercise 6: Pick the computer's move randomly

1. Import the random module at the top of your code so that you can use it.
2. Make a list of allowed moves where each move is lowercase text, e.g. "rock".
3. Choose a move randomly from the list you just made and assign it to the computer's move.

Remember to test your program!

When you run your program at this point it should:

- ✓ Print out the game title, rules and allowed moves.
- ✓ Ask the user what move they want to play.
- ✓ Pick a random move for the computer.
- ✓ Print out the human's and computer's moves.
- ✓ If the human wins, print out "You won!"
- ✓ If the computer wins, print out "You lost."
- ✓ If it is a tie, print out "You tied with the computer."

5. Playing again

At the moment, our program only runs once, and then it stops. But we want to play Scissors-Paper-Rock as many times as we like! We are going to change it in two ways. First, we're going to let the human enter how many games they want to play per round, and use a **for loop** to play that many games. Second, we're going to use a **while loop** to ask the user after each round if they'd like to play another round.

Firstly, what is a loop? Sometimes we want to do the same thing many times. We could just copy and paste the code lots of times, but that gets pretty boring. Instead, we can use a loop to do this for us. There are two types of loops in Python: `for` loops and `while` loops. A `for` loop will do something a certain number of times, whereas a `while` loop will do something repeatedly while some condition is true. Let's have a look at an example of a for loop.

```
for i in range(5):
    print("We're up to number", i)
```

The first line sets up how many times the loop will run. The code indented after this line is what will be run repeatedly. In this example, the loop will run 5 times and the variable `i` will give the number of iterations that we're up to. So, what will it print?

```
We're up to number 0
We're up to number 1
We're up to number 2
We're up to number 3
We're up to number 4
```

Wait, what? Why did it start at 0? This is because the `range()` function always starts at 0. This is a really handy function to know, you can also use it to set up lists of numbers quickly.

```
my_list = list(range(5))
print(my_list)
```

This will print

```
[0, 1, 2, 3, 4]
```

You can also run `for` loops over lists. It works much the same, but instead of the loop variable being a number, it is now an item in the list.

```
dinner_options = ["spaghetti", "tacos", "stir-fry"]
for option in dinner_options:
    print(option)
```

What do you think this will print?

Let's look at an example of a `while` loop.

```
keep_going = "yes"
while keep_going == "yes":
    keep_going = input("Keep going? ")
```

This time, instead of looping a certain number of times, our loop keeps going and going and going until its condition becomes false. This example creates a variable `keep_going` and initially sets it to `"yes"`. Then, the loop checks whether the variable is equal to "yes", which of course it is because we just set it to that, but then, inside the loop it asks the user if it should keep going. If the user says yes, then the loop condition is true and it loops again. If the user says no, then the condition becomes false and the loop ends.

You can also put loops inside other loops! What do you think the following code will do?

```
keep_going = "yes"
while(keep_going == "yes")
    for(i in range(5))
        print("We're up to number", i)
    keep_going = input("Keep going? ")
```

Notice that the bodies of both for and while loops need to be indented! OK, we now know everything we need to make the program play as many games as we like!

Exercise 7: Play a round of games

1. Define a variable to store the number of games the human wants to play per round.
2. Ask the human how many games they want to play per round. Think about:
    a. What command will you use? Remember you've used this before!

b. Where should the command go in your Python code?
c. Python will always store information from the `input()` function as text. To use your variable as a number, you need to change it using the command `int(<variable>)`.

3. Wrap your code in a for loop so that it plays as many games as the human entered. Think about:
   a. Where should your for loop start, and end?
   b. How will you indent your code?
   c. You will need a loop counter, how will this fit in your code? What variable will you use as your loop counter? Remember you can use the print statement to check the counter in your loop.
4. Print "Round over!" when the for loop exits.

Remember to test your program!

When you run your program at this point it should:

- ✓ Print out the game title, rules and allowed moves.
- ✓ Ask the user how many games they want to play per round.
- ✓ Ask the user what move they want to play.
- ✓ Pick a random move for the computer.
- ✓ Print out the human's and computer's moves.
- ✓ If the human wins, print out "You won!"
- ✓ If the computer wins, print out "You lost."
- ✓ If it is a tie, print out "You tied with the computer."
- ✓ Repeat the game as many time as the user said they wanted to play in a round.
- ✓ Print out "Round over!"

Exercise 8: Play many rounds of games

1. Define the variable you will use to exit the while loop *before* you begin the while loop. Think about:
    a. What should the initial value of this variable be?
2. Wrap your code in a while loop so that it keeps playing rounds until the user wants to stop. When wrapping your code in a while loop, think about:
    a. Where should the while loop start, and end?
    b. How should you indent your code?
3. Ask the user if they would like to play another round. Think about:
    a. When should you ask the user if they would like to play another round?
4. Print "Goodbye!" when the loop exits.

Tip! If your while loop keeps running and doesn't stop, this is called an infinite loop. You can stop an infinite loop by pressing CONTROL-C. If you see an infinite loop, your while loop is probably defined in the wrong place. Try defining it somewhere else!

When you run your program at this point it should:

- ✓ Print out the game title, rules and allowed moves.
- ✓ Ask the user how many games they want to play per round.
- ✓ Ask the user what move they want to play.
- ✓ Pick a random move for the computer.
- ✓ Print out the human's and computer's moves.
- ✓ If the human wins, print out "You won!"
- ✓ If the computer wins, print out "You lost."
- ✓ If it is a tie, print out "You tied with the computer."
- ✓ Repeat the game as many time as the user said they wanted to play in a round.
- ✓ Print out "Round over!"
- ✓ Ask the user if they want to play another round.

- ✓ If the user says “yes”, repeat another round of games.
- ✓ If the user says “no”, print “Goodbye!”

Congratulations! You’ve completed Scissors-paper-rock!! Now you can move on to the advanced workbook and start adding some extensions.