MOONHACK 2020

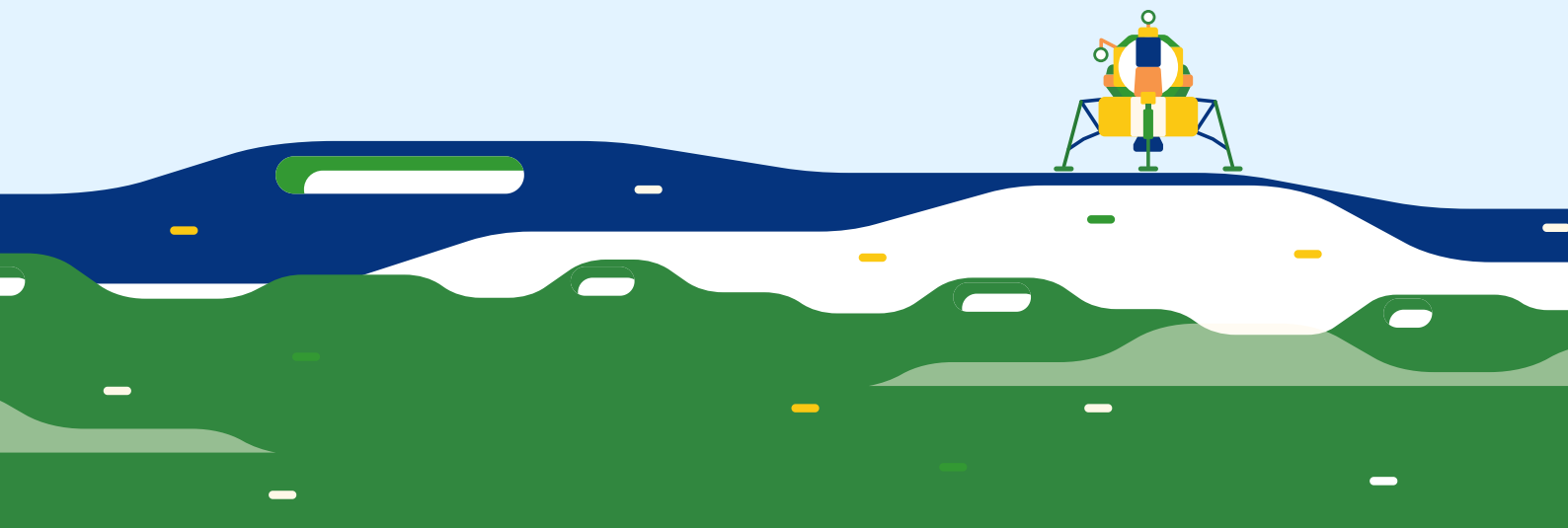
PYTHON WATER ON MARS

ENGLISH

BROUGHT TO YOU BY CODE CLUB AUSTRALIA
POWERED BY TELSTRA FOUNDATION

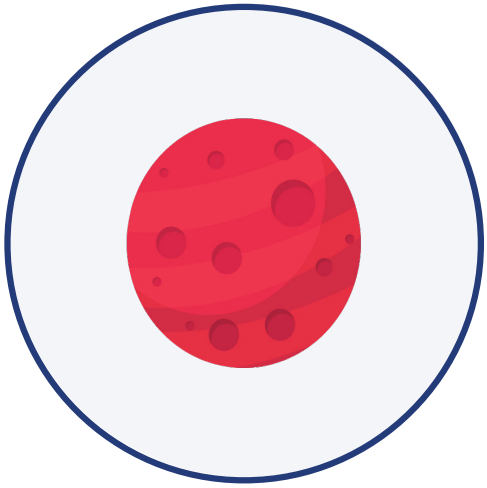
code club / AUSTRALIA | POWERED BY TELSTRA FOUNDATION

SUBMIT AND BE COUNTED AT
MOONHACK.COM



MOONHACK 2020

Water on Mars



Water is an incredibly precious resource, even in space! How much water do you need on Mars?

INTRODUCTION

What you will make

You will make a text-based survival game, in which the player must make water conscious decisions in order to survive for as long as possible without running out of water on Mars.

What you will learn

- Using and creating variables in text-based coding
-

What you will need

HARDWARE

A computer capable of accessing Trinket online

The starter code can be found here bit.ly/pythonwater

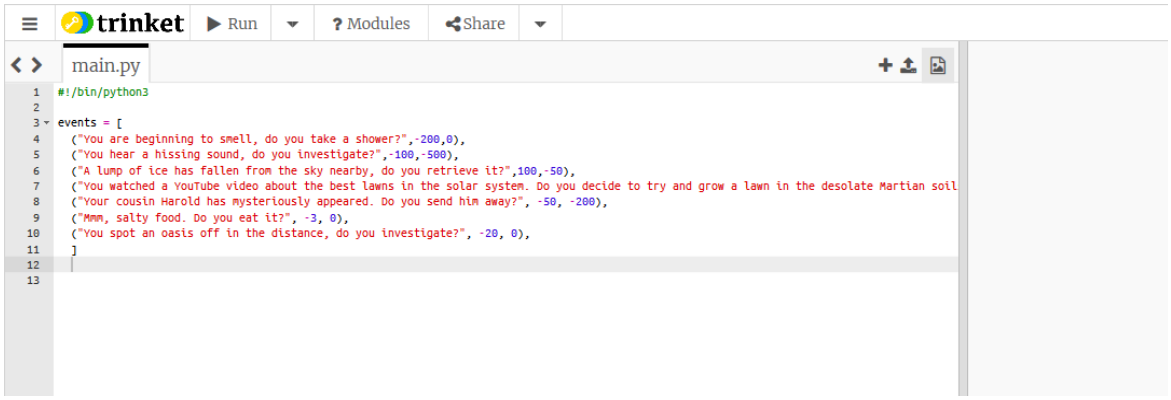
Additional notes for educators

Check out our blog post for this project with tips, curriculum and supporting material at medium.com/@codeclubau

1. KEEPING TRACK OF WATER

Let's set up a variable that we can use to keep track of how much water we have

- Open the starter project at bit.ly/pythonwater . Your window should look like this:



On the left is your code window; you will notice there is already some code here (we'll come back to that later). On the right is your execution window; this is where the output of your game will show. At the top is the run button; you will use this button to run your code.

We need to give our player some water to start off with. Let's start with a nice round number of 1000L.

Add the following highlighted code at the bottom of your program:

```
10    ("You spot an oasis
11    ]
12
13    water = 1000
```

We want our user to keep playing the game as long as they have water. When they run out of water, the game will end. Let's create our main game loop, which will continue as long as the user has water:

```
13    water = 1000
14
15    while water > 0:
```

Notice the colon (:) on the end of our line. This little character might seem insignificant, but if we forget it, Python will complain!

Next, let's add some water use to our main game loop. This will be a quantity that is subtracted every day, regardless of the player's choices, because the player needs to drink.

```
15 while water > 0:
16   water -= 5
```

Add the following code inside your loop:

Notice the two spaces in front of line 16 We call this an indent. This tells python that the instruction on that line is inside the loop. In other words, it happens every time we go around the main game loop (if you've used Scratch, it's similar to the orange loop blocks).

Now it's time for some output. Let's tell the user that the game is over when they run out of water. Add the following code outside of your main game loop (remember to get rid of the indent!):

```
15 while water > 0:
16   water -= 5
17
18 print ("Game Over :(")
```

Run your code. What happens?

Python will just try to run our code as quickly as possible. Without any output in the main loop, we have no way of knowing what's happening until we get to the end. Let's add some output to our main loop.

```
15 while water > 0:
16   water -= 5
17   print("A day has passed.")
18
19 print ("Game Over :(")
```

Run your code again. Now you should see all the days that pass before the game ends.

Let's change that last line of code that we inserted to tell us how much water we added. We can combine the value of our variables with the text that we're printing using the + sign.

```
15 while water > 0:
16   water -= 5
17   print("A day has passed. Your Water " + water + "L")
18
19 print ("Game Over :(")
```

Run your code again. Did you get an error?

A big part of programming is trying to work out why we're getting the errors we're getting. In this case, the error is telling us that we "cannot concatenate 'str' and 'int' objects". When programming, you will often have to work out what these strange error messages mean.

This error is telling us that we're trying to combine two different things that Python doesn't know how to combine, the text (string, shortened to str) and the number we're using to keep track of our water (integer, shortened to int). To fix this, we can tell Python to convert our integer to a string with the following code:

```
15 while water > 0:
16     water -= 5
17     print("A day has passed. Your Water " + str(water) + "L")
18
19 print ("Game Over :(")
```

Run your code. You should be getting an output telling us how much water we're using on each day, before the player's character runs out of water

Challenge: Change the daily water use

Currently, we're using 5 litres of water per day. What do you think is the absolute minimum a person needs to survive? Can you change the code so that the player uses that amount of water?

2. KEEPING TRACK OF DAYS

So far, we've kept track of the amount of water a player has used, but in order to see how well they've done, we're going to keep track of the number of days that have passed.

First, we're going to set the initial value of our day variable, starting at day 1.

```
13  water = 1000
14  day = 1
15
16  while water > 0:
```

Next, every day that passes, we want to increase the value of the current day by one. Add the following code to your main game loop:

```
16  while water > 0:
17    water -= 5
18    print("A day has passed. Your Water " + str(water) + "L")
19    day += 1
20
21  print ("Game Over :(")
```

Finally, we're going to let the player know how well they've done. We'll do this by replacing the final print statement with a statement that includes the number of days that have passed.

```
19    day += 1
20
21  print("you lasted " + str(day) + " days")
```

Notice that we had to use str(day) again, to convert the number into text.

Run your code. You should get an output for every day, and then a final output that says how many days the player has lasted.

Challenge: Output the day number every day

Currently, we're only using the day variable at the very end. Can you change the code to output the day number every day?

Currently, the output will say:

A day has passed. Your water 950L

Can you change it to say:

Day 10. Your Water 950L

3. YOUR DAILY EVENT

Now we're going to add some gameplay. Every day, we're going to give the player a choice, and that choice will determine how much water they lose or gain.

You've probably noticed that there is a whole bunch of code at the start that you didn't write. This is a list of events that could take place on Mars. Let's add another print statement to our main loop. This will access our events list and choose one at random.

```
18    print("A day has passed. Your Water " + str(water) + "L")
19    print(random.choice(events))
20    day += 1
```

Oh no! Another error. This one says “name ‘random’ is not defined”. That’s because random isn’t a part of python by default, it’s part of a module. We need to import it to be able to use it.

```
1  #!/bin/python3
2  import random
3
4  events = [
```

Run your code. Every day, you should have a random event being printed out.

We’ll need to use the event multiple times on the given day. If we were to use random.choice (events) each time we wanted to use the event, we’d get a different event! Instead, let’s store that event in a variable called event, which we can use multiple times without having to worry.

```
19  print("A day has passed. Your Water " + str(water) + "L")
20  event = random.choice(events)
21  print(event)
22  day += 1
```

Run your code again. You should get a similar result to the previous time.

You’ve probably noticed that the event in our output is surrounded by a whole bunch of extra information and symbols. That’s because we’re storing the question and values for the amount of water to gain or lose if the player answers yes or no in the same place.

We really only want to ask the player the question without showing them the extra information. To access the question, we can use what is called index notation. The index is just the position that the element sits at, starting at 0.

Change your print line to the following:

```
20  event = random.choice(events)
21  print(event[0])
```

Run your code again. That’s a lot better, but the player can’t respond to the question.

Instead of using a print statement, let’s use an input statement. This tells python to expect the user to write something.

```
20  event = random.choice(events)
21  input(event[0])
22  day += 1
```

Now, let's store that input in a variable, because we'll want to use it later.

```
20    event = random.choice(events)
21    response = input(event[0])
22    day += 1
```

Run your code again. Your game should now wait for the player to respond every day.

4. USING THE PLAYER'S ANSWER

The player is responding to the question, now we need to use their response to judge their water use.

The user should answer either yes or no to the question. If they answer "yes", we should add the first number in the event:

```
21    response = input(event[0])
22    if response == "yes":
23        water += event[1]
24    day += 1
```

Note that even though we're adding, a lot of the numbers in the events are negative. Just like in maths, adding a negative number is the same as subtracting the positive of that number.

If the player answers "no", we should add the second number in the event.

```
22    if response == "yes":
23        water += event[1]
24    elif response == "no":
25        water += event[2]
26    day += 1
```

Run your code. They player should now be able to answer the questions. You will notice that the amount of water that the player has will change according to their decisions. What happens if the player answers something other than yes or no?

5. VALIDATE USER INPUT

You now have a working game, but the player can answer any way they want. We only want to accept "yes" and "no" as answers.

There are only two options, yes and no, so we can prompt the user to answer the way we want them to answer. Modify your response line to the following:

```
20    event = random.choice(events)
21    response = input(event[0] + " (yes/no): ")
22    if response == "yes":
```

That's better, but the player can still answer any way they want. A player could enter something else in order to cheat the game, or mistype their answer. Let's add an "else" statement to catch any response other than yes or no.

```
24    elif response == "no":
25      water += event[2]
26    else:
27      print("Please answer yes or no!")
28    day += 1
```

Run your code. What happens if the player enters something other than yes or no?

We are now asking the user to correct their response, but then we're asking them the next question anyway! We need to keep asking them the same question until they give us an answer we want to hear. We can do this with a loop.

You can do this by highlighting the lines that you want to indent and pressing 'tab' on your keyboard. This adds all the highlighted text into the 'while True' loop you've just created.

```
20    event = random.choice(events)
21    while True:
22      response = input(event[0] + " (yes/no): ")
23      if response == "yes":
24        water += event[1]
25      elif response == "no":
26        water += event[2]
27      else:
28        print("Please answer yes or no!")
29    day += 1
```

Run your code. What happens if you answer something other than yes or no? It should be asking you the question again. But what happens if you do answer yes or no? It's still asking you the question again! Oh my!

The "while True" loop is a loop that will always keep looping. These are quite useful if you want to go forever, but we don't want to go forever, we only want to go until the user answers yes or no! Fortunately, we have a special command that can break us out of a loop, even if it's one that otherwise would go forever. That command is "break".

Let's add a break command when the player answers yes or no.

```
22      response = input(event[0] + " (yes/no): ")
23      if response == "yes":
24          water += event[1]
25          break
26      elif response == "no":
27          water += event[2]
28          break
29      else:
```

Run your code again. You should now only be getting the same question again if you don't answer yes or no.

Things are working well now, but there's one more tiny little issue that some players may come up against. What happens if, instead of answering "yes", the player answers "Yes" with a capital Y?

Python is case sensitive. That means that if you are comparing two strings, it will think they are different strings just because they have different capitalisation. Fortunately, this is easy to fix. Python has a command that you can use on strings to make them all lower case. Change your response line to the following:

```
21  while True:
22      response = input(event[0] + " (yes/no): ").lower()
23      if response == "yes":
```

Run your code. It should now be accepting "Yes", "YES", "yeS" and any other combinations of capitalisation.

Challenge: Add more events

There are a few different events, but can you add more?

Hint: if you have difficulty, try copying and editing one of the events that are already there.

Challenge: Earth game

Can you modify the game to be about Earth instead of Mars? Think about the kinds of events that might happen, and the choices that a player may have to face.

Advanced Challenge: Play it again, Sam

Can you give the player an option to keep playing after they've lost the game?

Congratulations you're a Moonhack changemaker!

Don't forget to talk to an adult about registering your participation at moonhack.com