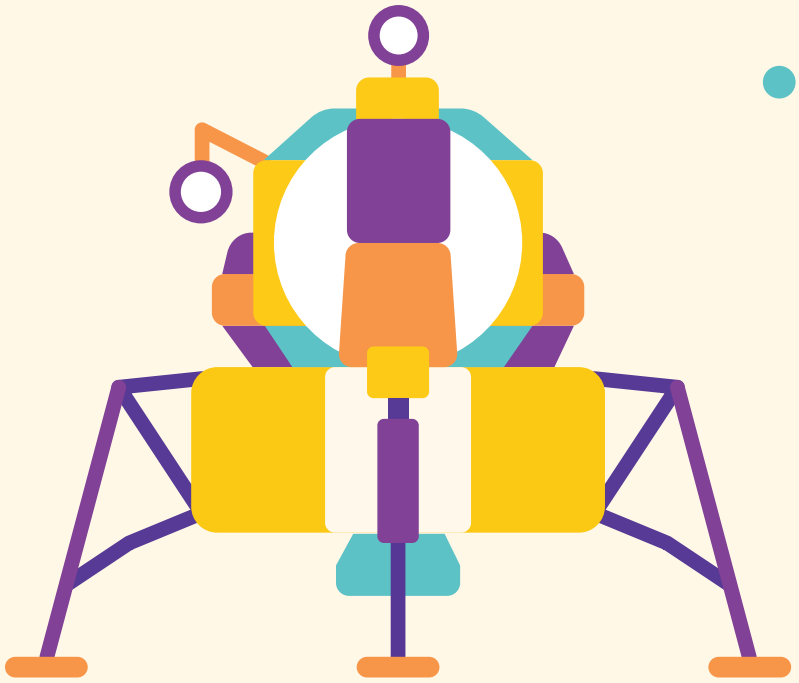
MOONHACK 2019

PYTHON PROJECT

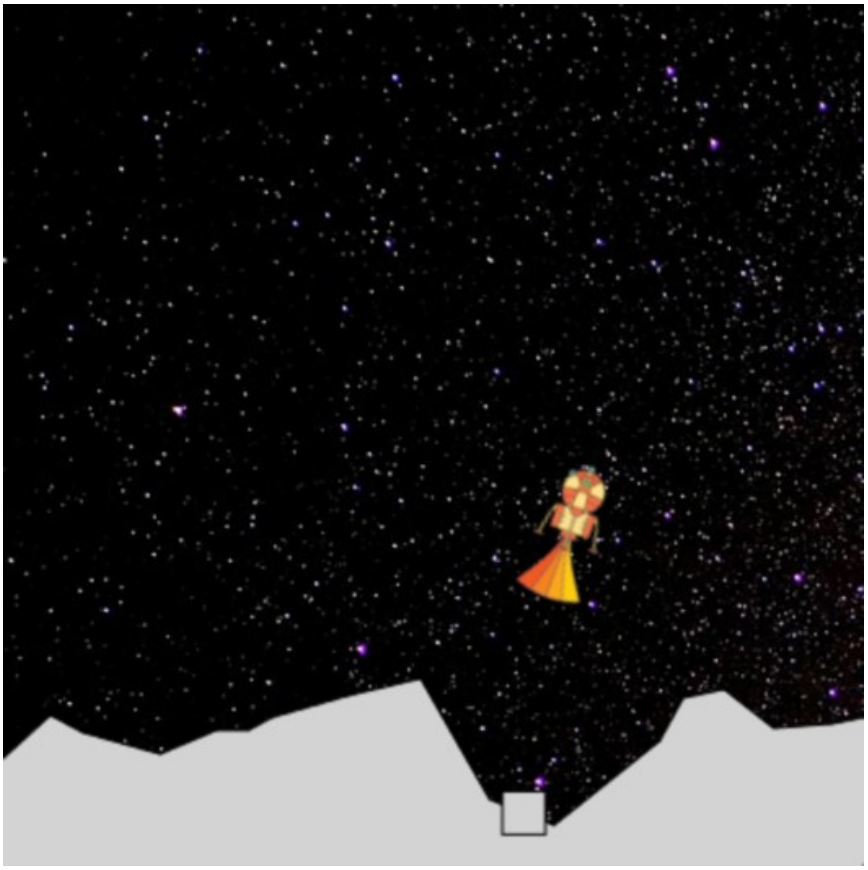


Introduction

On 20 July 1969, Neil Armstrong took the first steps on the moon. As we celebrate the 50th anniversary of this incredible achievement, we have a chance to reflect on the difficulty and audacity of every part of the Moon missions. In this project, we're going to simulate the moon landing by creating a Lunar Lander game in Python.

Instructions

The lander will fall, and you can use the arrow keys to control the angle and the space key to turn the thrusters on or off.



What you will need

Hardware

A computer with internet connection

Software

Trinket online editor

Step 1: Preparing the Lander

Activity Checklist

- [ ] Open the Python starter project by going to https://trinket.io/python/ac3342d0a8. Here you'll see three tabs: `main.py`, `terrain.py`, and `landerClass.py`. We will be coding in `main.py`; the other two tabs have been set up for you. If you press the ▶ Run button, it will randomly generate the lunar surface and place your landing pad. We will be trying to land on this landing pad.

- [ ] In your `main.py` tab, you will see that some code has already been added. These are `import` statements which bring in the code that we will need later on. We will add all of our code **after** these import statements.

- [ ] First, we need to create a lander. We can do this by create a new *instance* of the `landerClass`:

```
6 lander = landerClass()
```

▶ Run your program. You will see that the Lander module appears at the very top of the screen.

- [ ] Next, we want to control the lander using the keyboard. We can do this using the the `Screen().onkey()` command, which will register when we press certain keys. Let's start with using the left arrow to make our lander turn left:

```
7 Screen().onkey(lander.left,"Left")
```

The `onkey()` command takes two *arguments*: the first is the command that we want to execute (in this case, the `left` command has been set up as part of the `lander` class), the second is the key that we're looking out for (the `Left` key).

▶ Run your program. Does anything happen when you press the `Left` key?

- [ ] Nothing happens because we've told Python to respond to the `Left` key, but we haven't actually told it to listen out for the keys. We need to use the `listen()` command to fix this:

```
8 Screen().listen()
```

▶ Run your program. When you press the `Left` key, your lander should rotate to the left.

Note: If nothing still happens, make sure you're focused on the Result window, you can do this by clicking on the right hand side of your screen.

- [ ] Now we want to add code that will turn right when you press the `Right` key, and turn the thrusters on and off when we press the space key:

```
7  Screen().onkey(lander.left,"Left")
8  Screen().onkey(lander.right,"Right")
9  Screen().onkey(lander.toggleThrust,"Space")
10 Screen().listen()
```

▶ Run your program. You should now be able to control your lander.

Step 2: Making the Lander Fall

Activity Checklist

- [ ] We want to make our lander drop due to the lunar gravity. We will do this using a `while` loop. A `while` loop takes a `True` or `False` value, or another expression that *evaluates* to a `True` or `False` value. Add a loop that will continue forever:

```
11  while True:
```

- [ ] Next, we need to add the code that will actually make the lander fall. Without intervention, the speed at which the lander falls will increase due to the effect of the acceleration due to gravity, so we will add the fixed `ACCELERATION` amount to the `yVel` (y velocity) from the previous loop:

```
11  while True:
12      lander.yVel += lander.ACCELERATION
```

- [ ] If we run our code now, it still won't fall, because we haven't yet told it what to do with `yVel`. Every time we go through our loop, we need to tell it to subtract our `yVel` from the current position (we subtract because we're going down):

```
11  while True:
12      lander.yVel += lander.ACCELERATION
13      lander.sety(lander.ycor() - lander.yVel)
```

▶ Run your program. The lander should fall towards the ground at a slowly increasing rate (the increase will be quite slow, we are on the moon after all!)

- [ ] Right now, we're dropping right through the surface of the moon. We only want to continue moving until we reach the surface of the moon. To do this we will change our `while` loop, to use the `aboveLine()` function. The `aboveLine()` function is defined in the `terrain.py` file. It takes an `x` and a `y` coordinate, and returns true if that point is above the line, and false if it's below it. Modify your code to the following:

```
11  while terrain.aboveLine(lander.xcor(),lander.ycor()):
12      lander.yVel += lander.ACCELERATION
13      lander.sety(lander.ycor() - lander.yVel)
```

▶ Run your program. It should now fall until it reaches the surface, and then stop.

Step 3: Controlling the Lander

Activity Checklist

- [ ] Currently, we can turn on the thrusters, but they don't actually do anything. We will code in our while loop that will reduce our speed if the thrusters are activated:

```
13      lander.sety(lander.ycor() - lander.yVel)
14      if lander.thrusters:
15          lander.yVel -= lander.THRUST
```

▶ Run your program. When you turn your thrusters on, it will slow down your lander. If they're on long enough, your Lander will start going up.

- [ ] We're now successfully using our thrusters to slow our lander, but what happens if we turn our lander using the arrow keys? We're still only going up and down, but we want to go side to side as well. We need to split our thrust into an up-and-down (y-axis) component, and a side-to-side (x-axis) component, but how can work work that out? TRIGONOMETRY TO THE RESCUE! Update your code to the following:

```
14      if lander.thrusters:
15          heading = radians(lander.heading())
16          lander.yVel -= lander.THRUST*sin(heading)
17          lander.xVel += lander.THRUST*cos(heading)
```

▶ Run your program. Your lander will still only be going up and down! Oh no!

- [ ] So far, we've defined `xVel` (the x velocity), but we haven't actually used it to move our lander. Let's add a line of code that will use our `xVel` to move our lander along the x-axis:

```
13      lander.sety(lander.ycor() - lander.yVel)
14      lander.setx(lander.xcor() + lander.xVel)
15      if lander.thrusters:
```

▶ Run your program. You should now have a fully functioning lunar lander that moves on both the y and x axis.

Step 4: Step 4: Landing Safely... or not

Activity Checklist

- [ ] At the moment, when we finish our loop, our program just stops. We want to check `if` our lander has successfully landed on our pad. Add the following code to make sure our lander has ended up on the pad:

```
19  if terrain.onPad(lander.xcor()):
20      lander.landed()
```

- [ ] If we don't land on the pad, we have crashed. Add an `else` statement:

```
19  if terrain.onPad(lander.xcor()):
20      lander.landed()
21  else:
22      lander.crash()
```

▶ Run your program. If you end up near the landing pad, you will have landed successfully, otherwise you will crash and your lander module will fall apart.

- [ ] That's looking pretty good, but currently we will still have a successful landing even if we're coming in really fast. Let's add a condition to our if statement that tells us we've only landed successfully if we come in at under 2 units of speed:

```
19  if terrain.onPad(lander.xcor()) and lander.yVel < 2:
```

▶ Run your program.

Challenge: More crash conditions

Currently, the lander will crash if it's going too fast downwards, but there are other situations in which it might crash. Can you check whether the x velocity and and the heading of the lander (the direction in which is pointing) is within reasonable limits?
Hint: the `abs()` function takes a number and gives you the absolute value for it. This might be handy because the x velocity can be either negative (going left) or positive (going right).

Challenge: Mars Lander

NASA needs you! Research the gravitational force on Mars relative to the moon and change the `ACCELERATION` of your Lunar module to reflect this.