Coding Projects & Tips

We're going to finish off by looking at some interesting coding elements that you can incorporate into your own unique projects and code. In this section you'll find: code for creating a loading screen for your content, text animations using ASCII art, clever ways in which you can include animated sections at the command prompt, and even code that can track the International Space Station in real-time, while displaying the astronauts on-board and its current latitude and longitude.

There are also pages of common and language specific mistakes, to help you create error-free and effective code in the future.

Creating a Loading Screen

If you're looking to add a little something extra to your Python code, then consider including a loading screen. The loading screen is a short introduction, or piece of art, that appears before the main part of your code.

LOAD""

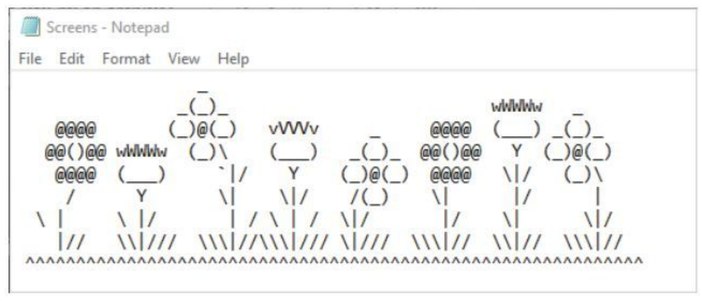
Back in the 80s, in the 8-bit home computing era, loading screens were often used to display the cover of a game as it loaded from tape. The image would load itself in, usually one-line-at-a-time, then proceed to colour itself in while the loading raster bars danced around in the borders of the screen.

Loading screens were a part of the package, and the buy-in for the whole game as an experience. Some loading screens featured animations, or a countdown for time remaining as the game loads, while others even went so far as to include some kind of playable game. The point being: a loading screen is not just an artistic part of computing history, but an introduction to the program that's about to be run.

While these days loading screens may no longer be with us, in terms of modern gaming we can still include them in our own Python content. Either just for fun, or to add a little retro-themed spice to the mix.

SCREEN$

Creating a loading screen in Python is remarkably easy. You have several options to hand: you can create a tkinter window and display an image, followed by a brief pause, before starting your main code, or you could opt for a console-based ASCII art version, that loads one-line-at-a-time. Let's look at the latter and see how it works.



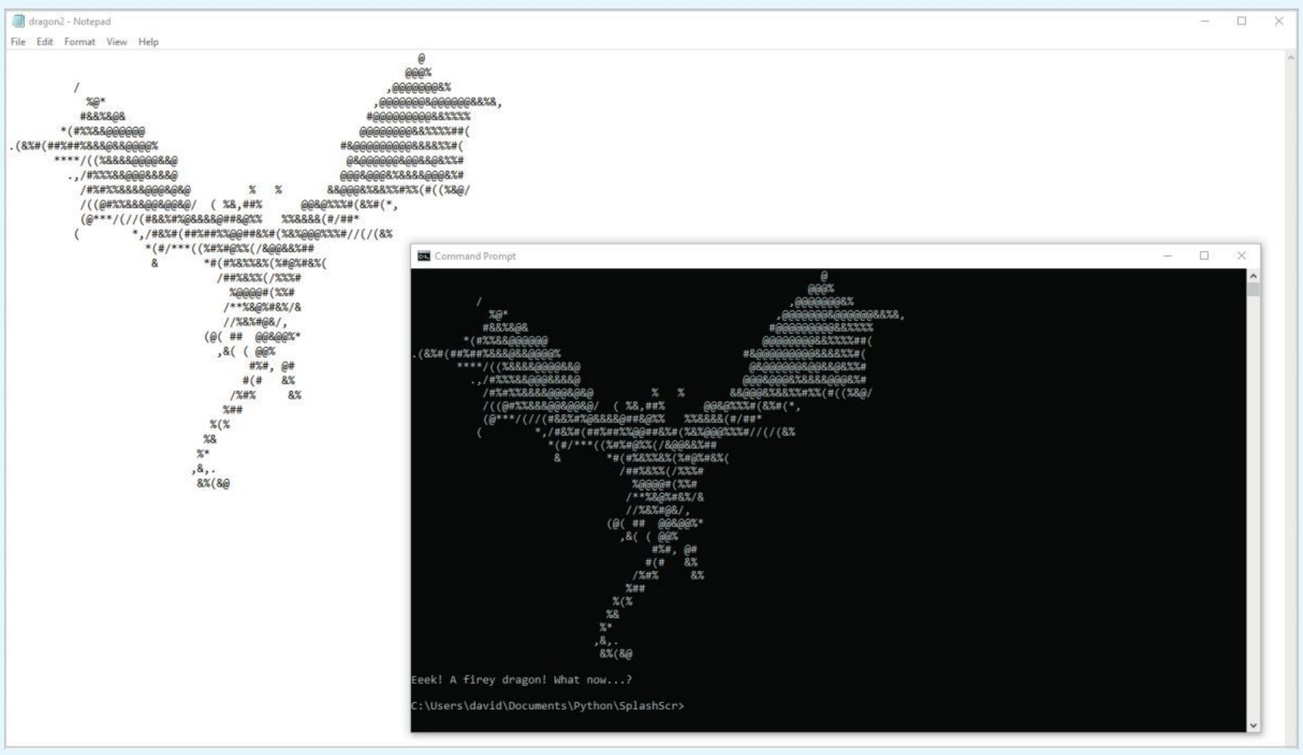
First you'll need some ASCII art, you can look up plenty of examples online, or use an image to ASCII Art converter to convert any images you have to ASCII. When you have your ASCII art, drop it into a newly created folder inside a normal text file.

Save the file, call it screens.txt for now.

ADVENTURE TIME

A good example of using loading screen ASCII art, text images, is when coding a text adventure. Once you've established your story, created the characters, events and so on, you could easily incorporate some excellently designed ASCII art to your game.

Imagine coming across a dragon, in game, and displaying its representation as ASCII. You can then load up the image lines one-by-one, and continue with the rest of the adventure code. It's certainly worth having a play around with and it'll definitely add a little something else extra.



THE CODE

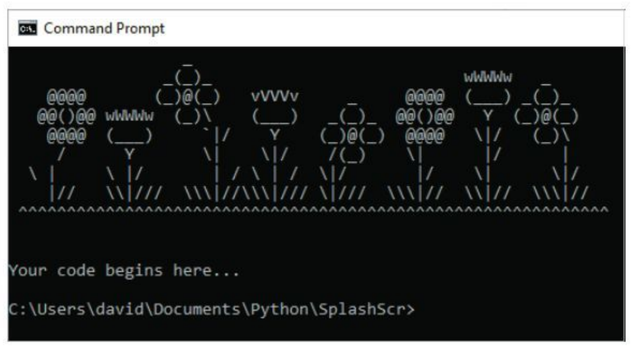
Launch Python and enter the following code to a New File:

```
import os
import time

def loading _ screen(seconds):
    screens=open("screens.txt", 'r')
    for lines in screens:
        print(lines, end='')
        time.sleep(seconds)
    screens.close()


#Main Code Start
os.system('cls' if os.name == 'nt' else 'clear')
loading _ screen(.5)

print ("\nYour code begins here...")
```

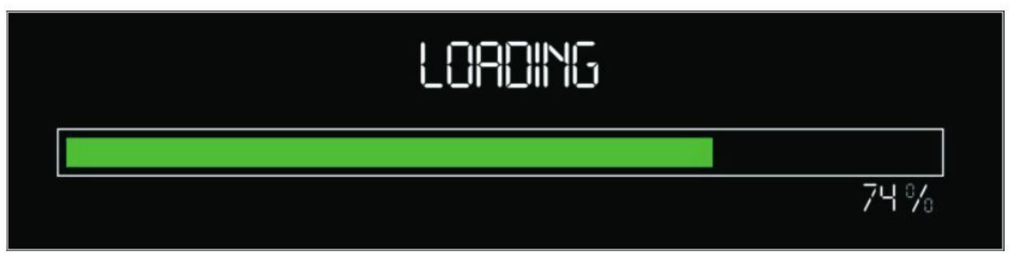
The code is quite simple: import the OS and Time modules and then create a Python function called loading_screen with a (seconds) option. Within the function, open the text file with the ASCII art as read-only and create a For loop that'll read the text file one-line-at-a-time. Next, print the lines – incidentally, the **lines, end=''** element will strip the newline from the text document, without it you'll end up with a double-line spaced display. Include the timing in seconds and close the text file buffer.

The final part of the code, #Main Code Start, is where you'll clear the screen (CLS for Windows, Clear for Linux and macOS) and call the function, together with the output number of seconds to wait for each line to be written to the screen – in this case, .5 of a second.

Save the code as screens.py, drop into a Command Prompt or Terminal and execute it. The screen will clear and your ASCII art inside the text file will load in line-by-line, creating a loading screen effect.




```
Loading...
```



Another favourite introduction screen is that of a simple loading animation, where the word loading is displayed followed by some characters and a percentage of the program loaded. While it may not take long for your Python code to load, the effect can be interesting.

Create a New File in Python and enter the following code:

```
import os
import time

def loading _ bar(seconds):
  for loading in range(0,seconds+1):
    percent = (loading * 100) // seconds
    print("\n")
    print("Loading...")
    print("<" + ("-" * loading) + (" " * (seconds
    loading)) + "> " + str(percent) + "%")
    print("\n")
    time.sleep(1)
    os.system('cls' if os.name == 'nt' else 'clear')


#Main Code Start
loading _ bar(10)

print ("\nYour code begins here...")
```

The code works in much the same way as the previous except, instead of reading from a text file, it's running through a for loop that prints Loading… followed by an animation of sorts, along with a percentage counter; clearing the screen every second and displaying the new results. It's simple, yes, but quite effective in its design.

COMBINING THE TWO

How about combining the two elements we've looked at? Let's begin with a Loading… progress bar, followed by the loading screen. After that, you can include your own code and continue your program. Here's the code:

```
import os
import time

def loading _ bar(seconds):
  for loading in range(0,seconds+1):
    percent = (loading * 100) // seconds
    print("\n")
    print("Loading...")
    print("<" + ("-" * loading) + (" " * (seconds-
    loading)) + "> " + str(percent) + "%")
    print("\n")
    time.sleep(1)
    os.system('cls' if os.name == 'nt' else 'clear')


def loading _ screen(seconds):
    screens=open("screens.txt", 'r')
    for lines in screens:
        print(lines, end='')
        time.sleep(seconds)
    screens.close()


#Main Code Start
loading _ bar(10)
os.system('cls' if os.name == 'nt' else 'clear')
loading _ screen(.5)

print ("\nYour code begins here...")
```

You can of course pull those functions up wherever and whenever you want in your code and they'll display, as they should, at the beginning. Remember to have the ASCII text file in the same folder as the Python code, or, at the **screens=open("screens.txt", 'r')** part of the code, point to where the text file is located.

Text Animations

There's a remarkable amount you can do with some simple text and a little Python know-how. Combining what you've already learned, we can create some interesting animation effects from the command line.

THE FINAL COUNTDOWN

Let's begin with some example code that will display a large countdown from ten, then clear the screen and display a message. The code itself is quite simple, but lengthy. You will need to start by importing the OS and Time modules, then start creating functions that display the numbers:

```
def one():
    print("""
   1111111
  1:::::::1
 1:::::::::1
 111:::::::1
    1:::::1
    1:::::1
    1:::::1
    1:::::1
    1:::::1
    1:::::1
    1:::::1
    1:::::1
    1:::::1
 111::::::111
 1:::::::::::1
 1:::::::::::1
 111111111111
   """)

def two():
    print("""
  222222222222222
 2:::::::::::::::22
 2::::::222222:::::2
 2222222       2:::::2
               2:::::2
               2:::::2
            2222::::2
       22222::::::22
     22::::::::222
   2:::::22222
 2:::::2
 2:::::2
 2:::::2          222222
 2::::::2222222:::::2
 2::::::::::::::::::2
 22222222222222222222
   """)

def three():
    print("""
  333333333333333
 3:::::::::::::::33
 3::::::33333::::::3
 3333333      3:::::3
                3:::::3
                3:::::3
     33333333:::::3
     3:::::::::::3
     33333333:::::3
                3:::::3
                3:::::3
                3:::::3
 3333333      3:::::3
 3::::::33333::::::3
 3:::::::::::::::33
  333333333333333
   """)
```

The code in its entirety can be viewed from within our Code Repository: **https://bdmpublications.com/ code-portal**, where you're free to copy it to your own Python IDLE and use it as you see fit. The end effect is quite good and it'll be worth adding to your own games, or presentations, in Python.

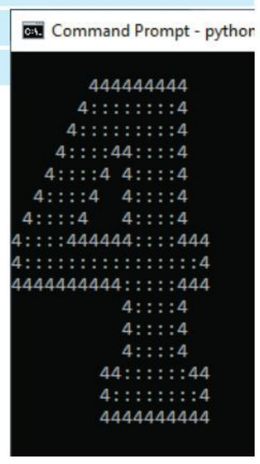
and so on to 10. It'll take some time, but it's worth it in the end. Of course, you can always take a different approach and design the numbers yourself.

The next step of the process is to initialise the code settings and start the countdown:

```
#Initialise settings
start = 10
message = "<       BLAST OFF!!            >"

#Start the countdown
for counter in range(start, 0, -1):
    if counter == 10:
        ten()
    elif counter == 9:
        nine()
    elif counter == 8:
        eight()
    elif counter == 7:
        seven()
    elif counter == 6:
        six()
    elif counter == 5:
        five()
    elif counter == 4:
        four()
    elif counter == 3:
        three()
    elif counter == 2:
        two()
    elif counter == 1:
        one()
    time.sleep(1)
    os.system('cls' if os.name == 'nt' else 'clear')
```

And finally, we can add a display for the message:

```
#Display the message
print("v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v")
print("<                                  >")
print(message)
print("<                                  >")
print("v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v")
print("\n\n\n")
```



To extend the code, or make it easier to use, you can always create the number functions in their own Python code, call it Count.py for example, then import Count at the beginning of a new Python file called Countdown.py, along with the OS and Time modules:

```
import os
import time
import count
```

From there, you will need to specify the imported code in the Countdown section:

```
#Start the countdown
for counter in range(start, 0, -1):
    if counter == 10:
        count.ten()
    elif counter == 9:
        count.nine()
    elif counter == 8:
        count.eight()
    elif counter == 7:
       count.seven()
    elif counter == 6:
        count.six()
    elif counter == 5:
        count.five()
    elif counter == 4:
        count.four()
    elif counter == 3:
        count.three()
    elif counter == 2:
        count.two()
    elif counter == 1:
        count.one()
```

This will pull the functions from the imported Count.py and print them to the screen.

ROCKET LAUNCH

Building on the previous countdown example, we can create an animated rocket that'll launch after the Blast Off!! message has been printed. The code for this would look something like this:

```
def Rocket():
  distanceFromTop = 20
  while True:
    os.system('cls' if os.name == 'nt' else 'clear')
    print("\n" * distanceFromTop)
    print("           /\         ")
    print("           ||         ")
    print("           ||         ")
    print("          /||\        ")
    time.sleep(0.2)
    os.system('cls' if os.name == 'nt' else 'clear')
    distanceFromTop -= 1
    if distanceFromTop <0:
      distanceFromTop = 20


#Launch Rocket
Rocket()
```

Here we've created a new function called Rocket, which produces the effect of an ASCII-like rocket taking off and scrolling upwards; using the distanceFromTop variable.

To use this, add it to the end of the previous countdown code and at the end of the Blast Off!! message, add the following lines:

```
print("\n\n\n\n")
input("Press Enter to launch rocket...")
```

This will allow your message to be displayed and then, when the user has hit the Enter button, the rocket will launch.

Again, the code in its entirety can be found in the Code Repository at: **https://bdmpublications.com/ code-portal**.

ROLLING DIE

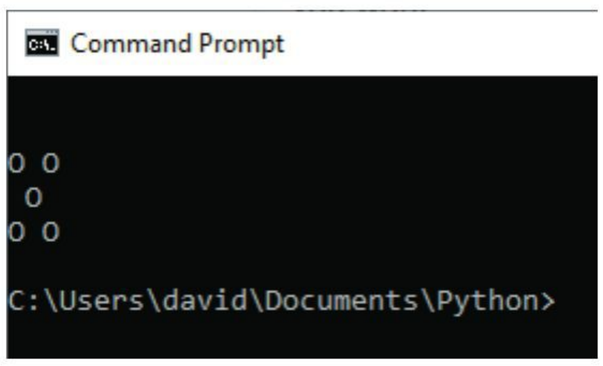
Aside from the rocket animation, together with its countdown, another fun bit of text-based animation is that of a rolling dice.

A rolling dice can be a great animation to include in an adventure game, where the player rolls to see what their score is compared to that of an enemy. The highest roller wins the round and the losers' health drops as a result. It's an age-old combat sequence, used mainly in the Dungeon and Dragons board games and Fighting Fantasy novels, but it works well.

The code you'll need to animate a dice roll is:

```
import os
import time
from random import randint

die     = ["   \n O \n    "]   #1
die.append("  O\n    \nO  ")   #2
die.append("O  \n O \n  O")    #3
die.append("O O\n    \nO O")   #4
die.append("O O\n O \nO O")    #5
die.append("O O\nO O\nO O")    #6

def dice():
  for roll in range(0,15):
    os.system('cls' if os.name == 'nt' else 'clear')
    print("\n")
    number = randint(0,5)
    print(die[number])
    time.sleep(0.2)

#Main Code Begins
dice()
```

You may need to tweak the O entries, to line up the dots on the virtual dice. Once it's done, though, you'll be able to add this function to your adventure game code and call it up whenever your character, or the situation, requires some element of luck, combat or chance roll of the dice.



Tracking the ISS with Python

Of the many amazing human achievements over the past decade or so, the International Space Station tops the bill. This incredible collaboration between nations sees vital experiments carried out in space as well as observations of our own planet.

TO BOLDLY GO...

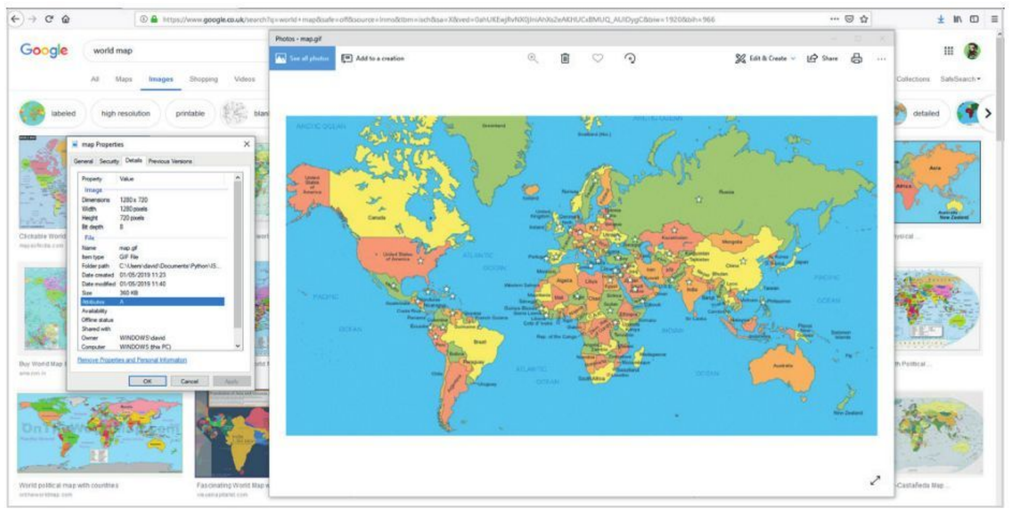
Indeed, the ISS is something most of us consider as a worthy example of what can happen when we work together. NASA, among other agencies, uses a wealth of Python code onboard the ISS to help automate routines, as well as act as an in-between link to translate code from one language to another, and then into a human-readable format. If you're considering a career in space, then learning Python is a must-have skill.

While we're not able to fill you in on all the details, regarding the code the ISS utilises, we can look at a fun Python script that will track the ISS, display the number of astronauts on board, update the station's current latitude and longitude every five seconds, while also displaying your current latitude and longitude.

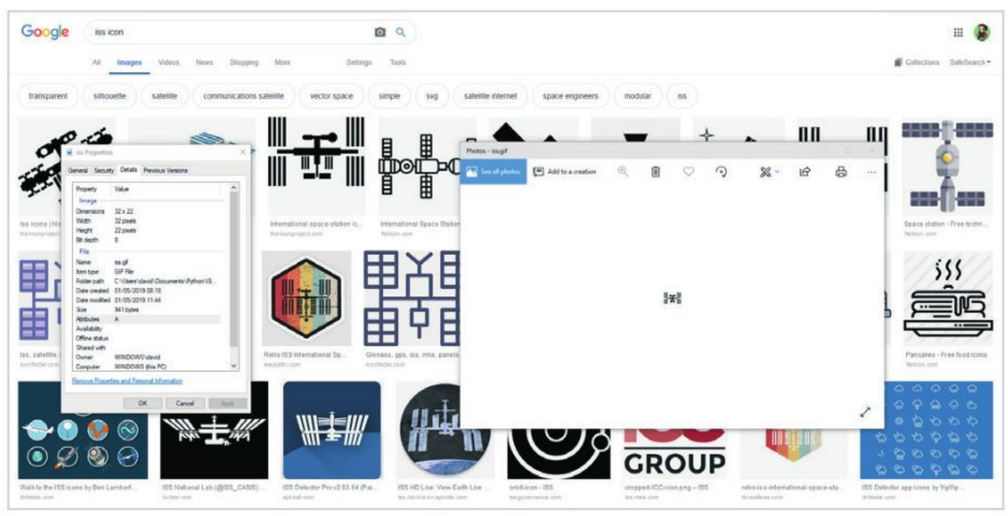
Displaying all that information in a single screen can become a little cluttered, so in this instance we're going to look at spreading all those details over three screens: a text document window -displaying the astronauts and your current latitude and longitude, a command line (or Terminal window) - displaying the continually updating latitude and longitude of the ISS as it orbits Earth, and a final, large window displaying a map of the world, together with an icon representing the ISS, that's updated as it orbits. Interested? Read on.

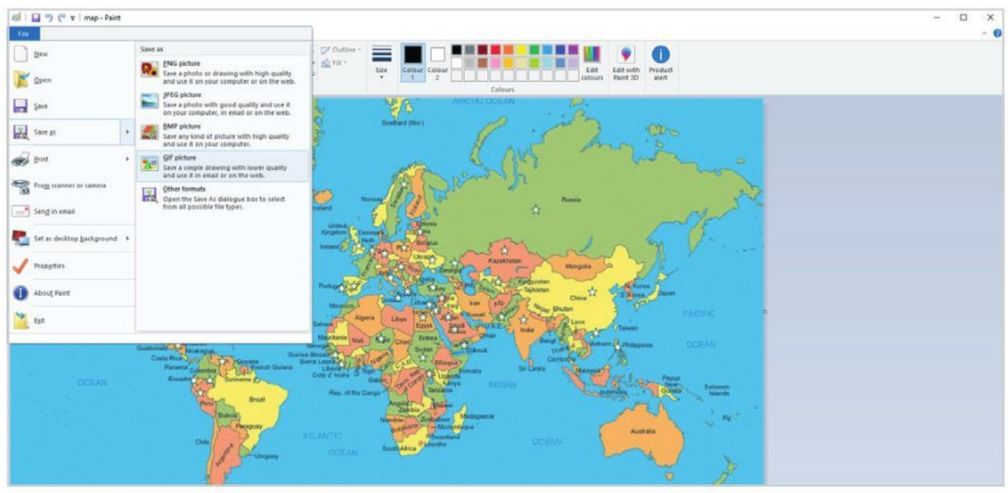
THE GRAPHICS

Firstly, we need to get hold of a map of the world, and an image of the ISS to use as an icon, that will be updated according to the position of the space station as it travels over the surface. A quick Google of World Map will help you out here. Look for one that's reasonably large, the one we used for this example was 1280 x 700, and one that has the names of the countries – if you're using this with young people, to help with putting shapes of countries to names.



Next, look for an ISS icon. As this is going to be a graphical representation of the location of the ISS, we need the image to be reasonably small so it doesn't drown out the locations on the map, but also prominent enough to see when the map is loaded. We opted for an image that's 32 x 22 pixels in size. Don't worry too much if you're not able to find one that small, you can always resize it in an image-editing app such as Paint or GIMP.





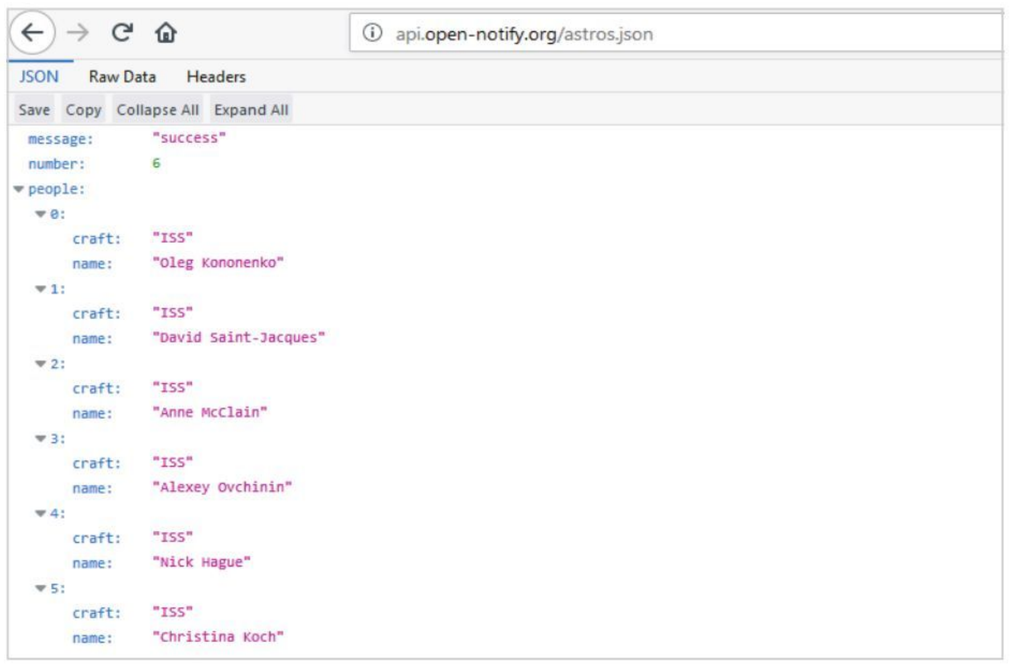
As we're going to be using Turtle, a component of tkinter, the downloaded images will need to be converted to GIF, since this is the default and recommended image format. You can easily look up a converter online, but using Paint in Windows 10, or GIMP, which is cross-platform, will suffice.

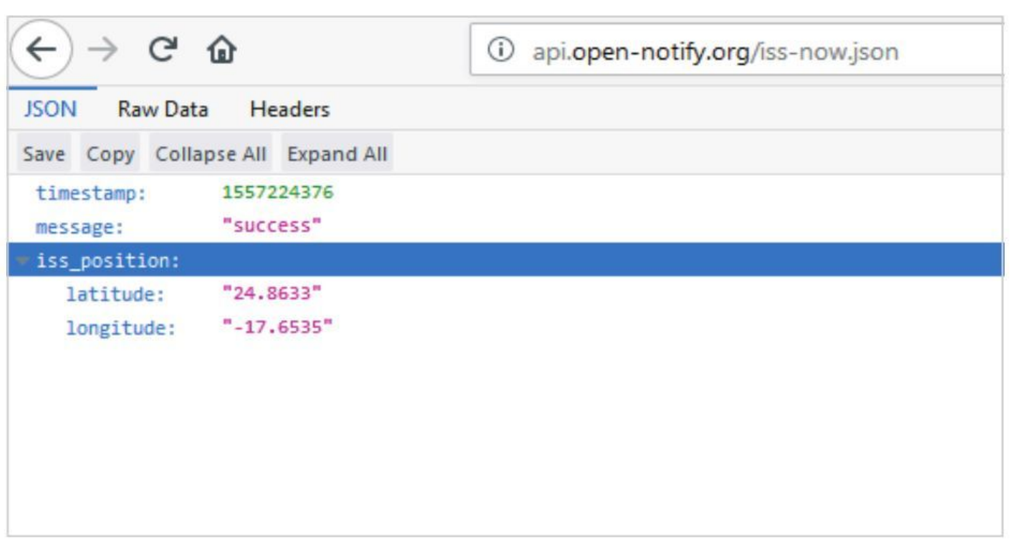
Simply load the image up in your image editor app and choose Save As, call them map and iss respectively, and click GIF as the image format. Remember to also resize the ISS image before saving it as a GIF.

THE CODE

The code we're using here utilises an open source API (Application Programming Interface) to retrieve real-time data online regarding the status of the ISS. An API enables applications to communicate with one another by providing the raw data that a programmer can pull out and interact with in their own code. In this case, the API in question is a web-based collection of raw data that's stored in a JSON (JavaScript Object Notation) format – an accessible and easy to use data-interchange interface.

In order for Python to interact with the JSON provided data, it needs to use the urllib.request and json modules. We're also going to be using a new module called geocoder, which will pull up a users' current latitude and longitude based on their IP address. The two JSON API's can be located at: **http://api.open-notify.org/astros.json**, and, **http://api.open-notify.org/iss-now.json**. One of which contains the data regarding the astronauts onboard the ISS, and the other contains the data regarding the current location of the ISS in longitude and latitude.





Let's begin by breaking the code into bite-sized chunks:

```
import json, turtle, urllib.request, time, webbrowser
import geocoder # need to pip install geocoder for your
lat/long to work.
```

First, we need to import the modules used in the code: json, turtle, urlib.request, time and webbrowser. The json and urlib.request modules deal with the data being pulled from the APIs, turtle will display the graphics, and time you already know. The webbrowser module will open text files in the default text file application – despite what its name may suggest. The geocoder module is a new element, and you will need to install it with: **pip install geocoder**. Geocoder can retrieve a users' location based on their IP address, as each ISP will have a geo-specific IP.

```
#Retrieve the names of all the astronauts currently on
board the ISS, and own lat/long - write to a file and display
url = "http://api.open-notify.org/astros.json"
response = urllib.request.urlopen(url)
result = json.loads(response.read())
a=open("iss.txt","w")
a.write("There are currently " + str(result["number"]) + "
astronauts on the ISS:\n\n")

people = result["people"]

for p in people:
  a.write(p["name"] + " - on board" + "\n")

g=geocoder.ip('me') # need to pip install geocoder, and
import as in the headers above
a.write("\nYour current Lat/Long is: " + str(g.latlng)) #
prints your current lat/long in the text file.
a.close()
webbrowser.open("iss.txt")
```

This section will use the json and urllib.request modules to pull the data from the API that contains the names of the astronauts onboard the ISS. It then creates a new text file called iss.txt, where it'll write 'There are currently X astronauts on the ISS…' and list them by name. The second part of this section will then use the geocoder module to retrieve your current latitude and longitude, based on your IP address, and also write that information to the end of the text file that contains the names of the astronauts. The last line, webbrowser.open("iss.txt") will use the webbroser module to open and display the newly written text file in the system's default text file reading app.

```
#Setup world map in Turtle
screen = turtle.Screen()
screen.setup(1280, 720)
screen.setworldcoordinates(-180, -90, 180, 90)
#Load the world map image
screen.bgpic("map.gif")

screen.register_shape("iss.gif")
iss = turtle.Turtle()
iss.shape("iss.gif")
iss.setheading(45)
iss.penup()
```

This section of the code sets up the graphical window containing the world map and the ISS icon. To begin we set up the turtle screen, using the resolution of the world map image we downloaded at the start of this tutorial (1280 x 720). The screen.setworldcoordinates syntax will

mark the boundaries of the screen, creating the x and y coordinates of the four corners of the canvas, so that the ISS icon can freely travel across the map of the world and wrap itself back to the opposite side when it reaches an edge. The ISS icon is set as the turtle pen shape, giving it an angle of 45 degrees when moving.

```
while True:
  #Load the current status of the ISS in real-time
  url = "http://api.open-notify.org/iss-now.json"
  response = urllib.request.urlopen(url)
  result = json.loads(response.read())

  #Extract the ISS location
  location = result["iss _ position"]
  lat = location["latitude"]
  lon = location["longitude"]

  #Output Latitude and Longitude to the console
  lat = float(lat)
  lon = float(lon)
  print("\nLatitude: " + str(lat))
  print("Longitude: " + str(lon))

  #Update the ISS location on the map
  iss.goto(lon, lat)
  #refresh every 5 seconds
  time.sleep(5)
```

Now for the final part of the code: Here we collect the location data from the ISS status API, pulling out the latitude and longitude elements of the JSON file. The code then prints the latitude and longitude data to the console/Terminal, transferring the data from a float to a string in order to print it correctly. The last section will use the latitude and longitude as variables lat and lon, to update the ISS icon on the map every five seconds.

Here's the code in its entirety:

```
import json, turtle, urllib.request, time, webbrowser
import geocoder # need to pip install geocoder for your
lat/long to work.

#Retrieve the names of all the astronauts currently on
board the ISS, and own lat/long - write to a file and display
url = "http://api.open-notify.org/astros.json"
response = urllib.request.urlopen(url)
result = json.loads(response.read())
a=open("iss.txt","w")
a.write("There are currently " + str(result["number"]) + "
astronauts on the ISS:\n\n")

people = result["people"]

for p in people:
  a.write(p["name"] + " - on board" + "\n")

g=geocoder.ip('me') # need to pip install geocoder, and
import
as in the headers above
a.write("\nYour current Lat/Long is: " + str(g.latlng)) #
prints
your current lat/long in the text file.
a.close()
webbrowser.open("iss.txt")

#Setup world map in Turtle
screen = turtle.Screen()
screen.setup(1280, 720)
screen.setworldcoordinates(-180, -90, 180, 90)
#Load the world map image
screen.bgpic("map.gif")
```

ISSTrack.py - C:\Users\david\Documents\Python\ISS Tracker\ISSTrack.py (3.7.0)

File Edit Format Run Options Window Help

```
import json, turtle, urllib.request, time, webbrowser
import geocoder # need to pip install geocoder for your lat/long to work.


#Retrieve the names of all the astronauts currently on board the ISS, and own lat/long - write to a file and display
url = "http://api.open-notify.org/astros.json"
response = urllib.request.urlopen(url)
result = json.loads(response.read())
a=open("iss.txt","w")
a.write("There are currently " + str(result["number"]) + " astronauts on the ISS:\n\n")

people = result["people"]

for p in people:
  a.write(p["name"] + " - on board" + "\n")

g=geocoder.ip('me') # need to pip install geocoder, and import as in the headers above
a.write("\nYour current Lat/Long is: " + str(g.latlng)) # prints your current lat/long in the text file.
a.close()
webbrowser.open("iss.txt")



#Setup world map in Turtle
screen = turtle.Screen()
screen.setup(1280, 720)
screen.setworldcoordinates(-180, -90, 180, 90)
#Load the world map image
screen.bgpic("map.gif")

screen.register_shape("iss.gif")
iss = turtle.Turtle()
iss.shape("iss.gif")
iss.setheading(45)
```

```
iss.penup()

while True:
  #Load the current status of the ISS in real-time
  url = "http://api.open-notify.org/iss-now.json"
  response = urllib.request.urlopen(url)
  result = json.loads(response.read())

  #Extract the ISS location
  location = result["iss_position"]
  lat = location["latitude"]
  lon = location["longitude"]

  #Output Latitude and Longitude to the console
  lat = float(lat)
  lon = float(lon)
  print("\nLatitude: " + str(lat))
  print("Longitude: " + str(lon))

  #Update the ISS location on the map
  iss.goto(lon, lat)
  #refresh every 5 seconds
  time.sleep(5)
```

```
screen.register _ shape("iss.gif")
iss = turtle.Turtle()
iss.shape("iss.gif")
iss.setheading(45)
iss.penup()

while True:
  #Load the current status of the ISS in real-time
  url = "http://api.open-notify.org/iss-now.json"
  response = urllib.request.urlopen(url)
  result = json.loads(response.read())

  #Extract the ISS location
  location = result["iss _ position"]
  lat = location["latitude"]
  lon = location["longitude"]

  #Output Latitude and Longitude to the console
  lat = float(lat)
  lon = float(lon)
  print("\nLatitude: " + str(lat))
  print("Longitude: " + str(lon))
```

```
  #Update the ISS location on the map
  iss.goto(lon, lat)
  #refresh every 5 seconds
  time.sleep(5)
```

Create a new folder in your system, called ISSTrack (for example), and place the two graphics, as well as the Python code itself.

RUNNING THE CODE

The code is best executed from the command line or Terminal. Clear your desktop, enter your command line and navigate to where you've saved the code plus the two graphics. Launch the code with either: python ISSTrack.py or Python3 ISSTrack.py (depending on your system, and what you've called the Python code).

The code will launch two extra windows together with the command line window you already have open. One will be the text file containing the named astronauts along with your current latitude and longitude, and the other will be the world map with the ISS icon located wherever the ISS is currently orbiting. The command line window will start scrolling through the changing latitude and longitude of the ISS.