Week 7: Python's Built-in Tools

Introduction

In this sheet we are going to look into two very useful built-in Python tools:

- 'venv' : Python's infrastructure for building your own virtual environment
- 'pdb' : Python's debugger tool

First, let's walk you through the steps of how to build your own Python environment and why that might be useful.

SECTION 1: Virtual Environments

What is a virtual environment?
A virtual environment in Python can be seen as a 'fresh slate' - a separate instance of Python on your computer that doesn't have all the downloads that you might have previously done (i.e 'numpy', 'pygame' etc). This instance sits in a folder on your computer, ready to be accessed, used or deleted easily. Python has a built in module named **'venv'** which helps us do all this.

Why would you use a virtual environment?

- If you have multiple versions of Python downloaded on your computer, i.e Python3.6 and Python 3.9, your computer will have a default version that it uses all the time. You might need to use a different version of python for a new project, however don't want to mess up all your other projects, so you can create a virtual environment with a **specific version of Python** installed.
- These virtual environments are easy to delete (just delete the folder they are in), so it is sometimes neater to work with.

- If a new project requires you to download loads of new Python packages, that you necessarily don't want saved onto your computer after the project, then a virtual environment is perfect as you can **install the packages you need** and then delete the environment once you are done and the packages are no longer on your computer (rather than having to do 'pip3 uninstall...').

TO DO:

Now, let's walk through how to set up a Python environment with a new Python package installed.

1. First, try running the 'face.py' file given to you in this week's zip folder. I am going to assume that you get the error 'No module cv2 found' as you have not used cv2 before and have not installed it. If you have you may have to just pretend for this one (or try to import a Python package that you don't have installed instead)!

Creating Our Virtual Environment

1. Open terminal and change directories to the ZIP folder provided to you today ('Week7').
2. To create a new virtual environment we use the following command:

```
python3 -m venv name
```

   Now, you should substitute 'name' for what you want to call the environment - it is common to call it 'venv'.

   If you wish to use a **different Python version** for your environment, then we can specify that in the command. For example, I have Python 3.6 and 3.9 downloaded and Python3.9 is my default. So, when I run the command above my environment uses Python 3.9. To set it as Python 3.6, I would run:

```
python3.6 -m venv name
```

3. You should now see a folder within the Week7 zip folder - this is your environment.

Activating Our Environment

Currently, our terminal in running our normal version of Python , we want to change that to our new empty version of Python.

1. First, run the command the following command. This will show you all the Python packages that you have installed onto your computer (for comparisons sake):

```
pip3 list
```

2. With your terminal in the directory of the Week7 zip folder, run the following command (with 'name' replaced with what you called you venv):

   OS / Linux:

```
source name/bin/activate
```

   Windows:

```
name\Scripts\activate.bat
```

   You will now notice that your terminal has changed slightly - you are now inside your virtual environment.

3. Now, run the 'pip3 list' command again. This should be a pretty small list, showing you the base packages that are installed in your environment - this is a good way to check to see if you are in the right environment.

Installing The Packages

Now, let's install the packages we need into this environment: cv2, numpy, matplotlib.

1. Run the following command to install cv2:

```
pip3 install opencv-python
```

   Be careful with copying and pasting these commands and it sometimes adds spaces in places it shouldn't.
   This should also install numpy for you (can do 'pip3 list' to check if you want).

2. Install matplotlib using the following command:

```
pip3 install matplotlib
```

Using Your Environment

The best way to run your Python code is straight from the terminal, while the environment is activated. Just run:

```
python3 face.py
```

This will hopefully bring up a face-detection program - note that this has just been given as an example program to test if we can import packages! **To exit the program, press esc or the space bar**.

If you get an error, don't worry too much, some computers don't let Python connect to your webcam easy. However, as long as you get no errors with importing the packages then your environment has worked!

If you get the error 'Abort trap: 6', then try activating your environment through terminal outside of VSCode or PyCharm and running your Python file from there!

If you wish to use your environment and VSCode, just activate your environment within the VSCode terminal. There are ways to do it with PyCharm and Jupyter-Notebook too, however I will not cover that here. A quick YouTube search will probably bring up exactly what you need.

Closing Your Environment

When in terminal, you may want to switch back to your normal system - the following command should do the trick:

```
deactivate
```

To **delete your environment** just simply delete the folder that you created at the start!

SECTION 2: Debugging

So far when trying to write code in Python you have probably had to use the 'print' statement loads to figure out why your code my not be working as well. Well, this is where Python's debugger tool comes in very handy and allows us to step 'inside' the code and see what our computer sees at a set line in the code.

First, we will look at how to use it, then there are some files of broken code that you should try to fix using this tool to get some practice in!

First, let's look at how to use the tool. Naturally, it is built into Python so you do not need to download anything!

The debugger is a Python package, named **pdb**, so all we need to do to let Python know we want to use it is import the package using the following *within a Python file (not a terminal command)*:

```
import pdb
```

Now, just doing this will not actually do anything in your code - we need to tell our computer where to stop in the code so we can investigate at this point. We can do this by using pdb's function **'set_trace'** which tells our computer to pause wherever we have called upon that function within our code.

Once we have called this, we will enter the 'debugger' mode. Inside this there are 4 main calls that will be useful to you:

- 'next': tells the debugger to run the next line
- 'step': run the next line, but if it is a function then step into that function
- 'continue': run the code either until the end or until the next 'set_trace' function call
- 'exit': takes you back to the command line

TO DO:

1. Create a Python file with the following code in:

```
import pdb
X = 0
pdb.set_trace()
X += 1
```

2. Run the file and notice the terminal change slightly to let you know that you're now in 'debugger' mode. Note that the line that is printed above is the **next line** that Python will run (it has not executed this). **When you are in debugger mode, you can run any Python code that you wish from the terminal.**
3. Call the line $print(X)$ - what do you expect this to return?
4. Call the line $locals()$ - what do you notice at the end of this?
5. Call the line 'next', this will run the next line in the code (this is the line printed in terminal where the last arrow is).
6. Now check confirm that the value of $X = 1$ using either of the methods above.

Now, let's slightly change our code so that we have a function and a call for that function. We are going to use the following code:

```python
import pdb

def AddOne(N):
    N += 1
    return N

X = 0
pdb.set_trace()
X = AddOne(X)
```

TO DO:

1. Replace your code with the code above
2. Run the code and follow the same process that you just did:
   (a) Print X and check it's value
   (b) Use 'next' to run the next line
   (c) Check X's value again
   (d) Either use 'exit' or 'continue' to return to the command line
3. Do the same process that you just did, however use **'step' instead of next**. Keep running this command until you are inside the function. When inside the function call the 'locals()' command - this should show you the locals variables within that function, so we should see N instead of X now. *Note that you can use 'globals()' to get the global variables still.*

TO DO:

1. Now, replace your code with the following:

```python
import pdb

X = 0
for i in range(3):
    pdb.set_trace()
    X += 1
```

2. Run the code and use the 'continue' command (instead of step and next) to go through the code, checking the value of X each time, making sure it is doing what it is supposed to.
3. Try using the command 'list' and see what it does.