

SECTION 16: WORKING WITH IMAGES WITH PYTHON - 24 minutes, 4 parts

2/4 Working with Images with Python

• overview

- -> working with images in Python
- -> the Pillow library <- the Python imaging library
- -> this has function calls
- -> **this is a library for image processing in Python which needs installing**

• installation

- -> pip install pillow <- in the terminal
- -> there is documentation for this

- -> there are three notebooks for this section
- -> there are multiple different image files which we are going to use in the ipynb file
- -> installing pillow

• data rate limits

- -> the data rate can be exceeded
- -> there is a default data rate limit
- -> if this is exceeded, then we restart it
- -> more information about this is on Stack Overflow



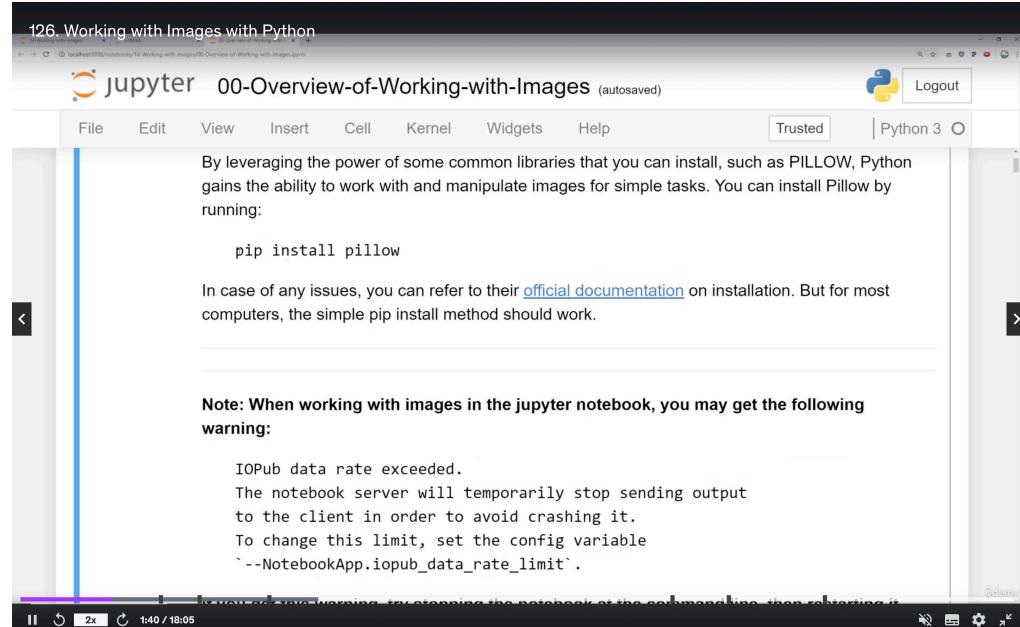
Complete Python Bootcamp

- In this lecture we will explore how to work with Images with Python.
- We will use the Pillow library for this, which is a fork of the PIL (Python Imaging Library) with easy to use function calls.
- We will need to install this additional library.



Complete Python Bootcamp

- Install it at your command line with:
 - **pip install pillow**
- You can find the official documentation for it at:
 - **pillow.readthedocs.io**



126. Working with Images with Python

jupyter 00-Overview-of-Working-with-Images (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

By leveraging the power of some common libraries that you can install, such as PILLOW, Python gains the ability to work with and manipulate images for simple tasks. You can install Pillow by running:

```
pip install pillow
```

In case of any issues, you can refer to their [official documentation](#) on installation. But for most computers, the simple pip install method should work.

Note: When working with images in the jupyter notebook, you may get the following warning:

```
IOPub data rate exceeded.  
The notebook server will temporarily stop sending output  
to the client in order to avoid crashing it.  
To change this limit, set the config variable  
`--NotebookApp.iopub_data_rate_limit`.
```

• opening images with Python

○ import modules

- -> from PIL import Image <- importing the Image class from the PIL module / library

○ open the image and store it in a variable

- -> he then defines another variable to open an image
- -> he prints returns the type of file
- -> this is a plugin
- -> there are multiple different file types

○ process the image

- -> he prints out one of the image files in the ipynb file
- -> we have an ipynb file, and multiple image files
- -> he has opened one of the image files in the ipynb file and stored it in a variable
- -> now he is printing out that variable <- returning the image
 - -> this can be done by typing the name of the variable which stores the image
 - -> or by its_name.show()
- -> then printing out more information associated with it
 - -> its size
 - -> the file name

The screenshot shows a Jupyter Notebook interface with the title "126. Working with Images with Python". The notebook contains a single cell with the following text:

By leveraging the power of some common libraries that you can install, such as PILLOW, Python gains the ability to work with and manipulate images for simple tasks. You can install Pillow by running:

```
pip install pillow
```

In case of any issues, you can refer to their [official documentation](#) on installation. But for most computers, the simple pip install method should work.

Note: When working with images in the jupyter notebook, you may get the following warning:

```
IOPub data rate exceeded.  
The notebook server will temporarily stop sending output  
to the client in order to avoid crashing it.  
To change this limit, set the config variable  
`--NotebookApp.iopub_data_rate_limit`.
```

At the bottom of the screen, there is a toolbar with various icons for navigating the notebook.

The screenshot shows a Jupyter Notebook interface with the title "126. Working with Images with Python". The notebook contains several cells of code and their outputs:

```
In [3]: from PIL import Image  
In [4]: mac = Image.open('example.jpg')  
In [5]: type(mac)  
Out[5]: PIL.JpegImagePlugin.JpegImageFile  
In [7]: mac.size  
Out[7]: (1993, 1257)  
In [ ]:
```

At the bottom of the screen, there is a toolbar with various icons for navigating the notebook.

The screenshot shows a Jupyter Notebook interface with the title "126. Working with Images with Python". The notebook contains several cells of code and their outputs:

```
In [5]: type(mac)  
Out[5]: PIL.JpegImagePlugin.JpegImageFile  
In [7]: mac.size  
Out[7]: (1993, 1257)  
In [8]: mac.filename  
Out[8]: 'example.jpg'  
In [9]: mac.format_description  
Out[9]: 'JPEG (ISO 10918)'  
In [ ]:
```

At the bottom of the screen, there is a toolbar with various icons for navigating the notebook.

126. Working with Images with Python

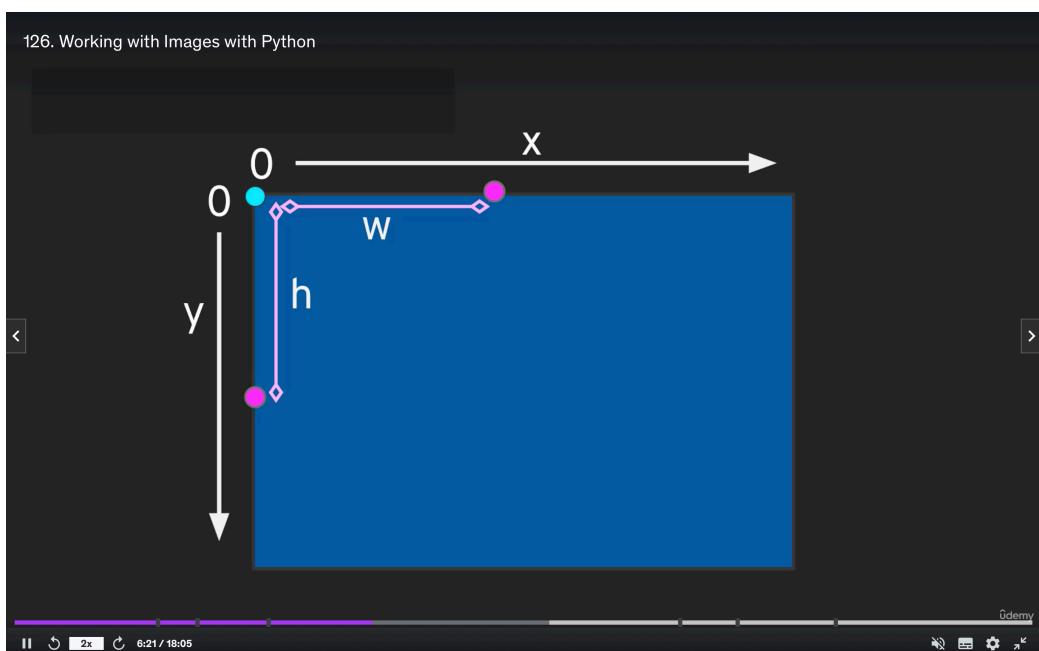
```

File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3
Out[9]: mac.filename
Out[9]: 'example.jpg'

In [10]: mac.format_description
Out[10]: 'JPEG (ISO 10918)'

Cropping Images
In [11]: mac.crop()

```



- -> cropping images
 - -> variable_name.crop()
 - -> this returns coordinates
 - -> this by the second quadrant on a x/y graph
 - -> on this coordinate, you can create a rectangle and crop out an image
 - -> this is done with variable_name.crop
 - -> the name of the variable which stores the image
 - -> pencil example

126. Working with Images with Python

```

File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3
Cropping Images
In [12]: mac.crop((0,0,100,100))
Out[12]:

```

```

In [13]: pencils= Image.open('pencils.jpg')
In [14]: pencils
Out[14]:

```

- o -> he has imported a pencil image and stored it in a variable
- o -> then returned the variable
- o -> he sets the x and y coordinates (stores values for these in two variables)
- o -> then sets the width and height which we want for the crop
- -> mac example
 - o -> he prints out the imported image of a mac in the ipynb file
 - o -> then the size of this, using .size
 - o -> **then to crop it, he defines the aspect ratio which we want as a function of this height**
 - o -> he sets the x and y coordinates which we want to crop
 - o -> knowing these requires a trial and error

126. Working with Images with Python

```
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
Out[15]: (1950, 1300)

In [18]: # BOTTOM PENCILS
x = 0
y = 1100

w = 1950 / 3
h = 1300

In [19]: pencils.crop((x,y,w,h))

Out[19]:
```

126. Working with Images with Python

```
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [21]: mac.size
Out[21]: (1993, 1257)

In [22]: halfway = 1993/2

In [23]: x = halfway - 200
w = halfway + 200

In [24]: y = 800
h = 1257

In [25]: mac.crop((x,y,w,h))
Out[25]:
```

126. Working with Images with Python

```
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [26]: computer = mac.crop((x,y,w,h))

In [27]: mac.paste(im=computer,box=(0,0))

In [28]: mac
Out[28]:
```

126. Working with Images with Python

```
In [3]: from PIL import Image
In [4]: mac = Image.open('example.jpg')
In [5]: type(mac)
Out[5]: PIL.JpegImagePlugin.JpegImageFile
```

In []: mac

→ copying and pasting images

- he stores an image in a variable
- then uses the .paste method with the image as its argument
- he's copied an image and pasted it over the top of another one
- he's set the coordinates on the second image where he wants the first one pasted
- multiple images can be copy and pasted over the same one

126. Working with Images with Python

```
In [31]: mac.size
Out[31]: (1993, 1257)
```

In [33]: mac.resize((3000,500))
Out[33]:

In [34]: mac.rotate(90)
Out[34]:

126. Working with Images with Python

Color Transparency

RGBA - Red , Green, Blue, Alpha

```
In [41]: red = Image.open('red_color.jpg')
In [42]: red
Out[42]:
```

→ the resize method

- we can print the size of an image
- the image is stored in the name of a variable
- the .resize method is used on this, with the argument the dimensions of the new image which we want

- > you can also use `variable_name.rotate`

The screenshot shows a Jupyter Notebook interface with the title "126. Working with Images with Python". In the top cell (In [43]), the code `blue = Image.open('blue_color.png')` is run. In the second cell (In [47]), the code `blue.putalpha(128)` is run, which changes the image's transparency. The resulting image (Out[48]) is a solid blue square with some transparency visible at the edges.

- > **colour transparency**

- > images have `rgba` <- `a` is alpha
- > these go from 0-255
- > alpha is 0, then the image is transparent
- > 255 is opaque

- > **changing alpha**

- > we have an example of an image which is stored in a variable

- > using `.putalpha(n)` on this variable can be used to transform the transparency of the image

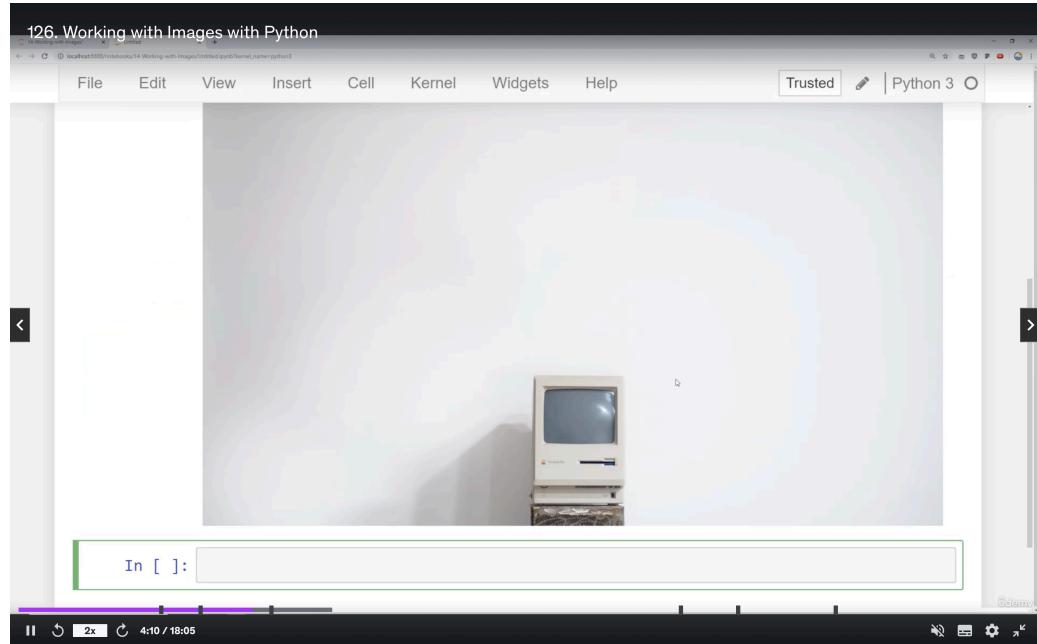
- > `n` is the value of alpha

The screenshot shows a Jupyter Notebook interface with the title "126. Working with Images with Python". A solid blue square is displayed. In the first cell (In [15]), the code `red.putalpha(128)` is run, creating a semi-transparent red rectangle overlaid on the blue square. The resulting image (Out[18]) shows the red rectangle with transparency.

The screenshot shows a Jupyter Notebook interface with the title "126. Working with Images with Python". A solid red rectangle is shown. In the first cell (In [19]), the code `blue.paste(im=red, box=(0,0), mask=red)` is run, pasting the red rectangle onto the blue background. The resulting image (Out[20]) is a solid purple square, indicating full transparency where the red and blue pixels met.

- -> **mixing two colours together**

- -> pasting one image on top of another
- -> this is done with the .paste method
- -> one of the images goes on top and the other on the bottom
- -> the arguments are the current image object, the box (alignment) and the mask (this should be the same as the image)
- -> the image we want on top, then how we want to align it based on the top left, then the mask being the same as the image
- -> knowing the image and where you want to place them
- -> to save the resulting image
 - ▶ -> the .save method
 - ▶ -> the argument of this should be the path where we want the image to be saved
 - ▶ -> the image can't already exist elsewhere



```
In [21]: blue.save("purple.png")
```

```
In [22]: purple = Image.open("purple.png")
```

```
In [23]: purple
```

The screenshot shows a Jupyter Notebook with three code cells. The first cell contains the command `blue.save("purple.png")`. The second cell contains the command `purple = Image.open("purple.png")`. The third cell displays the output `purple` with a preview of a solid purple rectangular image. The notebook has a dark theme. The title bar says "126. Working with Images with Python". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. A toolbar at the bottom shows a play button, a stop button, a zoom icon (2x), and a timer (17:53 / 18:05). The status bar at the bottom indicates "Trusted" and "Python 3".