CMPT 120

Lecture 27 – Computer Vision
Python – Creating a module and
List Comprehension

Last Lecture

- We *almost* solved the **image processing** problem of combining (merging) one image onto another
- In doing so, we were introduced to …
  - The **PIL** library and the **image** module
  - How to open an **image file**
    - And get information about the image file
      - like its **width** and **height**
  - How to read (`load`) the content of an **image file**
    - **Pixels** expressed as tuples -> (r,g,b)
    - **RGB** colour scheme -> color picker app.
  - How to go through each pixel of A?
    - **Nested `for` loops** are useful for traversing 2D data structures (or lists of lists)

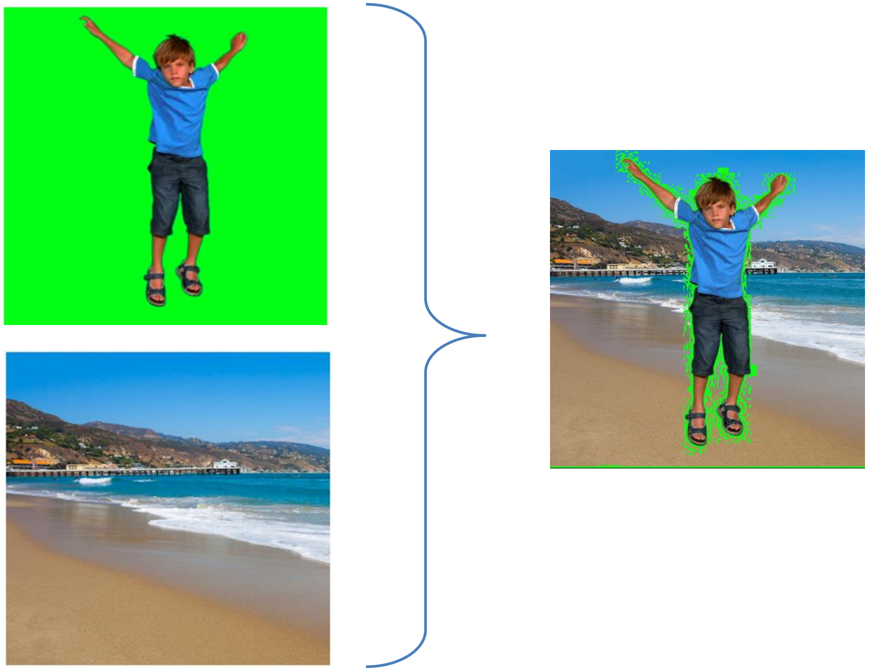
Today's Menu

- Continue having fun processing images
- Create our own **modules**
- Let's have another look at **Lists**!

Back to our "combining images" problem …

Step 1 - Problem Statement

- Combine (merge) the image file **`kid-green.jpg`** with the image file **`beach.jpg`** such as to produce an image file that displays the kid on the beach!



Back to our "combining images" problem …

Step 2 – Design

- Let's have a look at the rest of the comments in the `CombinedImages.py` program

Step 3 – Implementation

- Let's translate these comments into Python code keeping in mind the following questions:
  1. How to go examine each pixel of A?
  2. How to figure out if this pixel is green?
  3. If so …
     1. How to find the corresponding pixel in B?
     2. How to write the pixel in B into A?

Review - Two ways to access a pixel tuple's rgb values

```
# Way 1 - Get aPixel at (0,0)
aPixel = imageKidGreen[0,0]

# Get this pixel’s r value
r = aPixel[0]
# Get this pixel’s g value
r = aPixel[1]
# Get this pixel’s b value
r = aPixel[2]
```

Review - Two ways to access a pixel tuple's rgb values

```
# Way 2 - Get this pixel’s r value directly
g = imageKidGreen[0,0][0]
# Get this pixel’s g value directly
b = imageKidGreen[0,0][1]
# Get this pixel’s b value directly
b = imageKidGreen[0,0][2]
```

Create our own image function

Step 1 – Problem Statement

Write a function that returns **`True`** when a pixel is **green** and **`False`** otherwise

Step 2 – Design

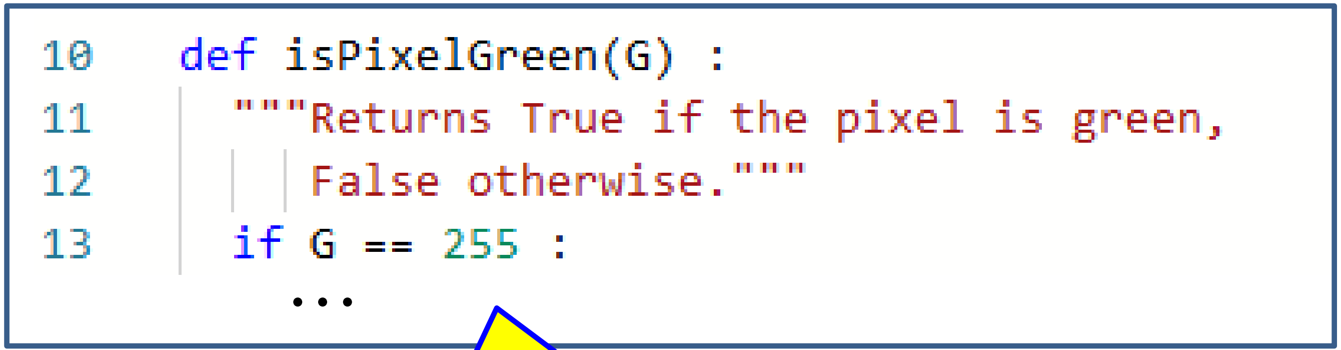
- How to discover if a pixel is **green**
  - Various ways of doing this:

```
1. if g == 255:
2. if g > 230 and g <= 255:
3. if r < 180 and r >= 0 and
        g > 230 and g <= 255 and
        b < 120 and b >= 0:
```

Where do these figures come from?

Let's give this function a try!

Step 3 – Implementation



```python
def isPixelGreen(G) :
    """Returns True if the pixel is green,
       False otherwise."""
    if G == 255 :
        ...
```

Can you complete the function?

Let's create our own module

- Since a module contains functions that are related to each other, perhaps we can create our own module **`myColourModule.py`** and put our functions
  - **`isPixelGreen(G)`**
  - **`ColourOfPixel(pixel)`** – from Practice Exam #7

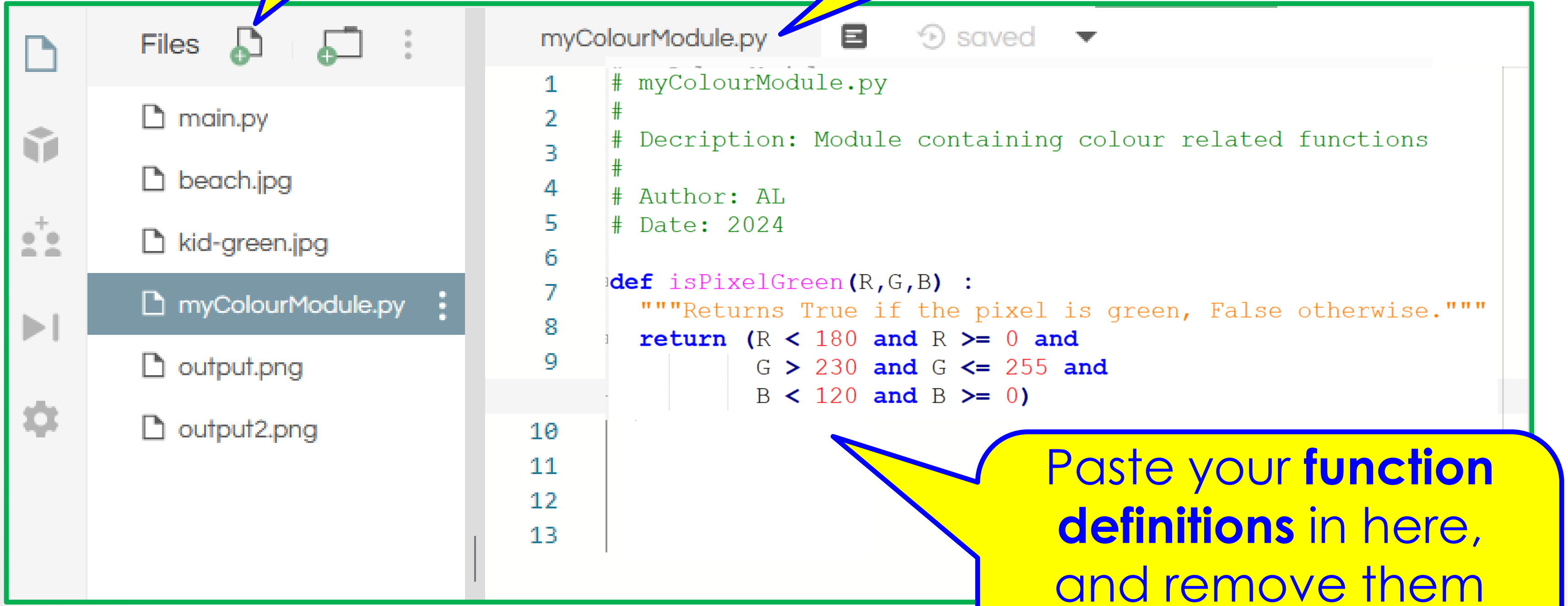
  into this module!

- **Description**: Module containing colour related functions

Let's create our own module

Create a new file

Name your **module** using a descriptive **filename** and a **.py** extension

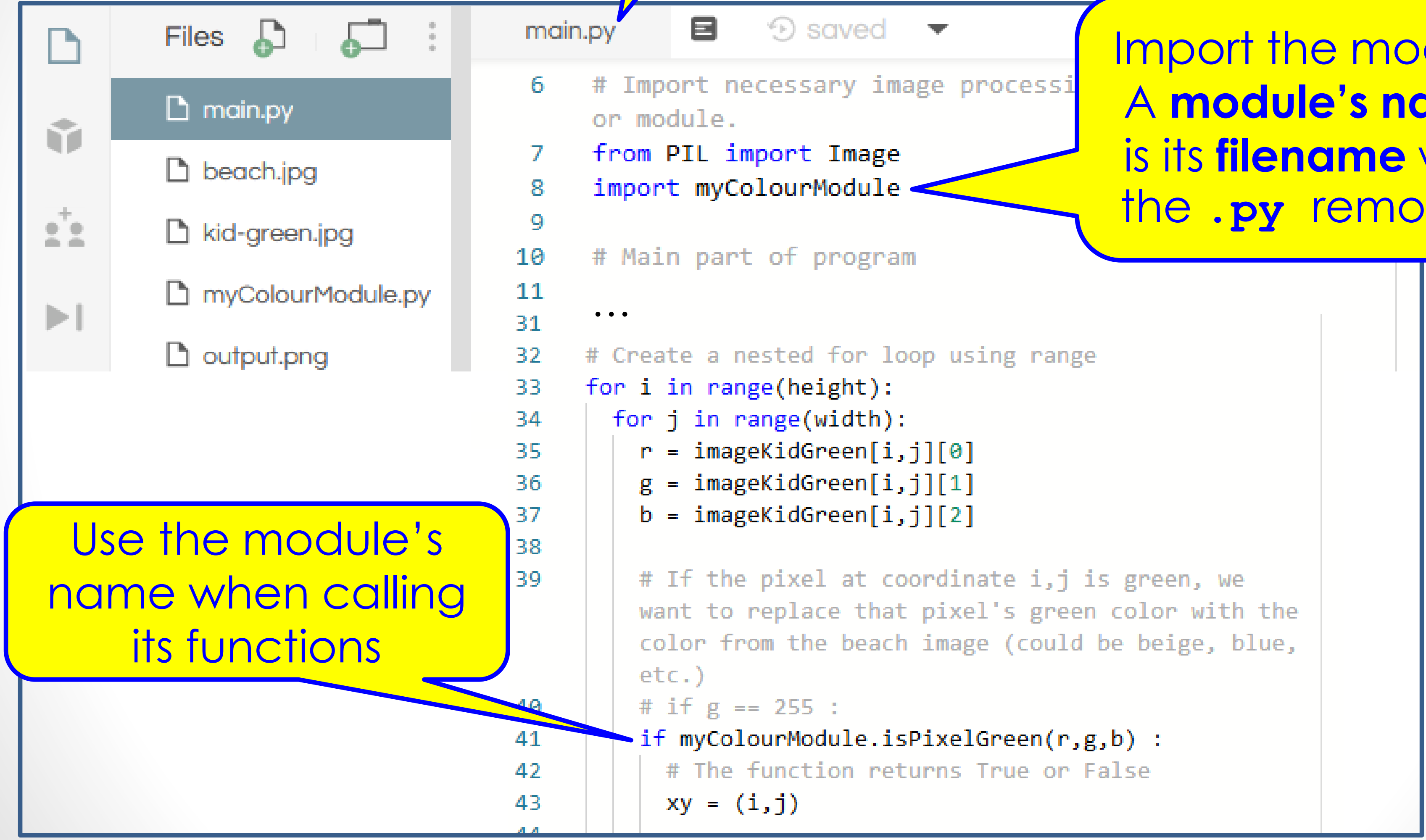


```python
# myColourModule.py
#
# Decription: Module containing colour related functions
#
# Author: AL
# Date: 2024

def isPixelGreen(R,G,B) :
    """Returns True if the pixel is green, False otherwise."""
    return (R < 180 and R >= 0 and
            G > 230 and G <= 255 and
            B < 120 and B >= 0)
```

Paste your **function definitions** in here, and remove them from your main program

Let's use our own Module!

In the main program



```
# Import necessary image processi
or module.
from PIL import Image
import myColourModule

# Main part of program

...
# Create a nested for loop using range
for i in range(height):
  for j in range(width):
    r = imageKidGreen[i,j][0]
    g = imageKidGreen[i,j][1]
    b = imageKidGreen[i,j][2]

    # If the pixel at coordinate i,j is green, we want to replace that pixel's green color with the color from the beach image (could be beige, blue, etc.)
    # if g == 255 :
    if myColourModule.isPixelGreen(r,g,b) :
      # The function returns True or False
      xy = (i,j)
```

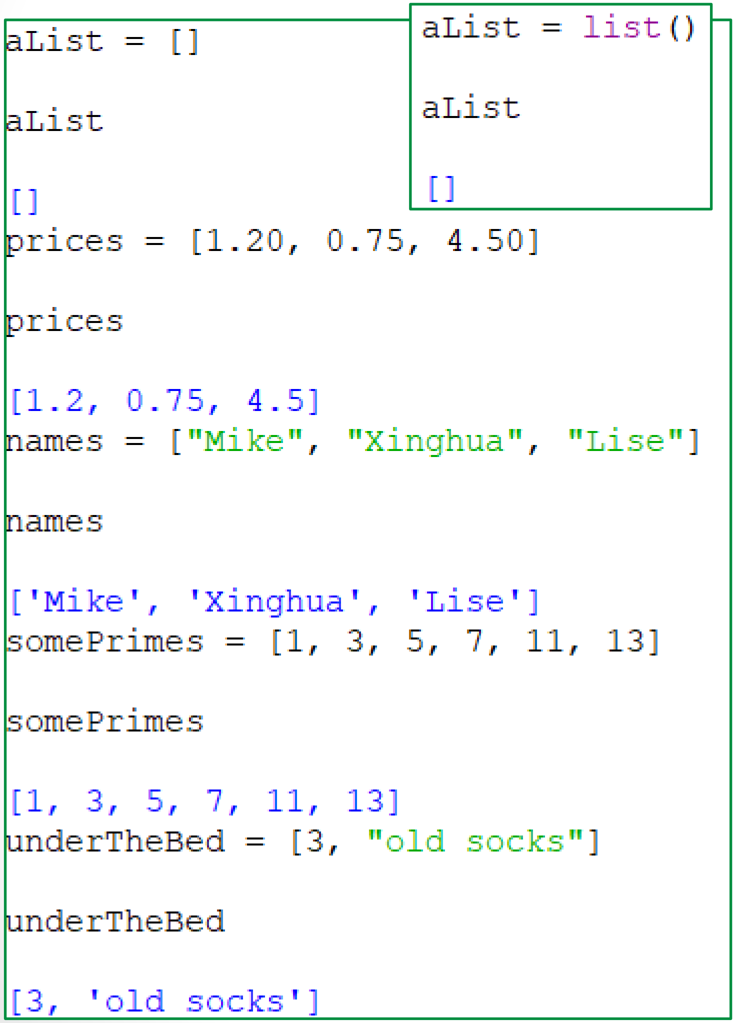
Import the module
A **module's name** is its **filename** with the `.py` removed

Use the module's name when calling its functions

Let’s have another look at **lists**

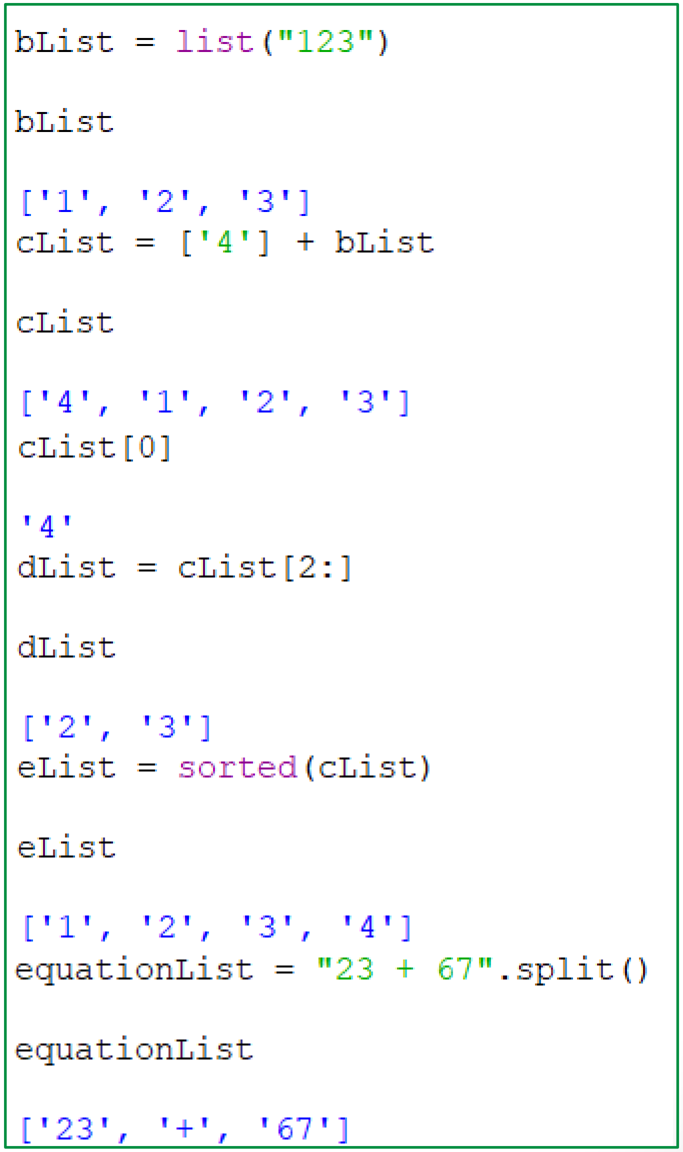
Review - Lists – so far …

At the IDLE shell:



```
aList = []

aList

[]
prices = [1.20, 0.75, 4.50]

prices

[1.2, 0.75, 4.5]
names = ["Mike", "Xinghua", "Lise"]

names

['Mike', 'Xinghua', 'Lise']
somePrimes = [1, 3, 5, 7, 11, 13]

somePrimes

[1, 3, 5, 7, 11, 13]
underTheBed = [3, "old socks"]

underTheBed

[3, 'old socks']
```

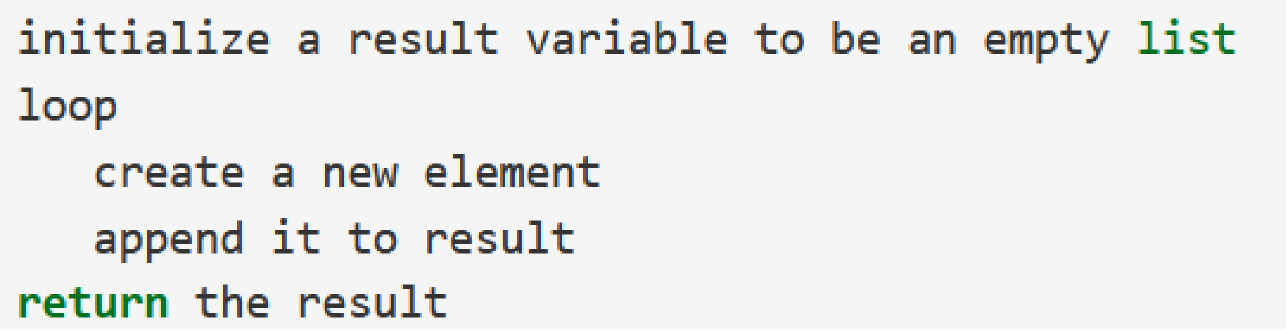
```
aList = list()

aList

[]
```



```
bList = list("123")

bList

['1', '2', '3']
cList = ['4'] + bList

cList

['4', '1', '2', '3']
cList[0]

'4'
dList = cList[2:]

dList

['2', '3']
eList = sorted(cList)

eList

['1', '2', '3', '4']
equationList = "23 + 67".split()

equationList

['23', '+', '67']
```

Review - Creating a list by accumulation

Algorithm:



```
initialize a result variable to be an empty list
loop
   create a new element
   append it to result
return the result
```

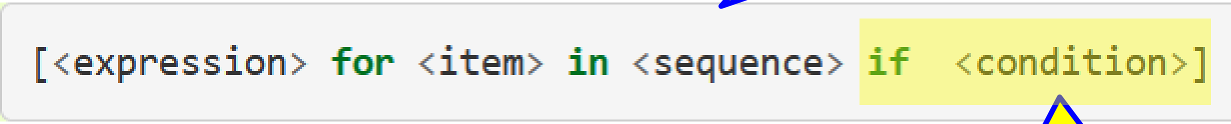
Another way of creating a list

List comprehension

- Concise way of creating a list

The expression within [ ] describes each element of the list that is being built.

- Syntax:



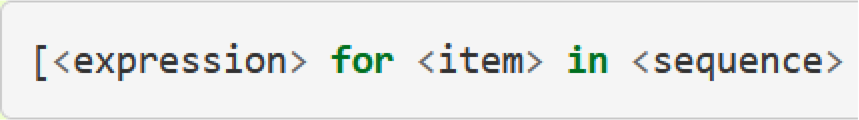
```
[<expression> for <item> in <sequence> if  <condition>]
```

Optional

How it executes?

1. The **for** loop (clause) iterates through each **item** **in** the **sequence**.
2. The items are filtered by the **if** clause if there is one.
3. The **expression** is executed for each **item** **in** the **sequence** (or each iteration of the **for** loop) …
4. … creating the resulting list.

List comprehension - Examples



```
[<expression> for <item> in <sequence>
```

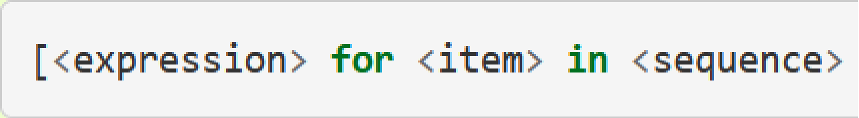
Example 1:

```
max = 5
list1 = ["*" for number in range(max)]
```

How it executes?

1. The **for** loop (clause) iterates through each **number in** the **sequence** -> 0, 1, 2, 3, 4.
2. The **expression** is executed for each iteration of the **for** loop … -> 5 times
3. … creating the resulting list -> `list1 = ['*', '*', '*', '*', '*']`

List comprehension - Examples



```
[<expression> for <item> in <sequence>
```

Example 2:

```
length = 4
list2 = [number for number in range(length)]
```

Example 3:

```
operandList = ["4", "5"]
operandList = [int(operandList[i]) for i in
                                        range(len(operandList))]
```

List comprehension – Examples

Example 4:

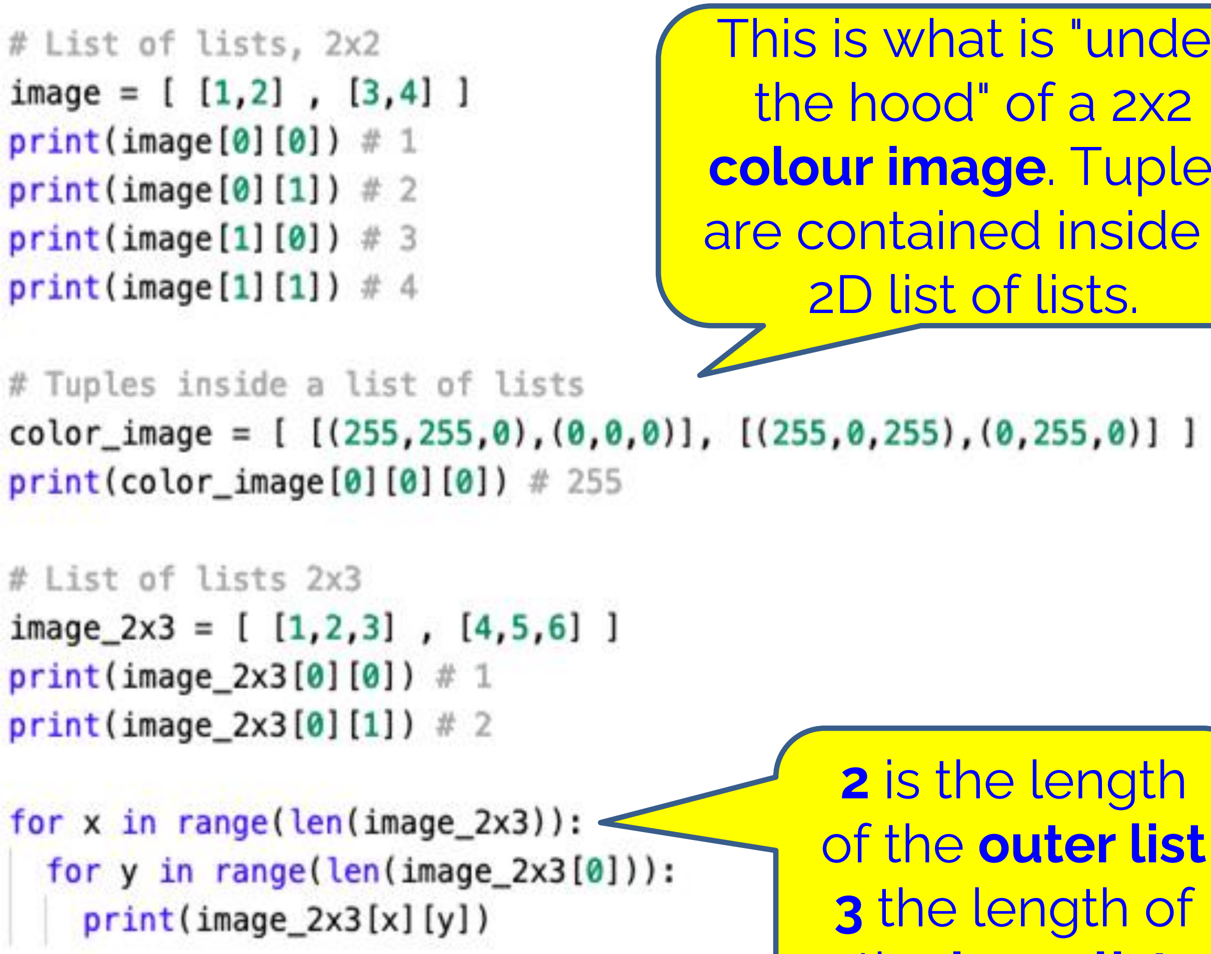
- How to create **a grid**

```
# Set variables
row = 5
column = 3
cellContent = " - "

# Create a grid
grid = [[cellContent for aColumn in range(column)]
                     for aRow in range(row)]
```

Review - Understanding images: 2D Data Structure

This is the Python syntax to access a list inside a list, i.e., a list of lists!

Note: it's slightly different than image access syntax, i.e., **image[c, r]**



```
# List of lists, 2x2
image = [ [1,2] , [3,4] ]
print(image[0][0]) # 1
print(image[0][1]) # 2
print(image[1][0]) # 3
print(image[1][1]) # 4

# Tuples inside a list of lists
color_image = [ [(255,255,0),(0,0,0)], [(255,0,255),(0,255,0)] ]
print(color_image[0][0][0]) # 255

# List of lists 2x3
image_2x3 = [ [1,2,3] , [4,5,6] ]
print(image_2x3[0][0]) # 1
print(image_2x3[0][1]) # 2

for x in range(len(image_2x3)):
  for y in range(len(image_2x3[0])):
    print(image_2x3[x][y])
```

This is what is "under the hood" of a 2x2 **colour image**. Tuples are contained inside a 2D list of lists.

2 is the length of the **outer list**
3 the length of the **inner list**

Next lecture

- **Practice Exam 8**
- Bring your **paper**, **pens/pencils**/**eraser**!