CMPT 120

Lecture 29 – Computer Vision
Problem Solving Strategies

Last Lectures

- Solved our **combining image file** problem
- Created our own **modules**
- Had another look at **Lists**!
  - List comprehension (single and nested)

- Practice Exam 8

Today's Menu

- We continue our exploration of **Computer vision**
  - We shall develop a **jellybean computer vision program** using the following concepts:
    - Problem-solving Strategies
    - Using append to add to a list
    - Multiplying and dividing in Python with * and /
    - String formatting when we output float values
  - We shall recycle the idea of colour functions we created last time

Jellybean Quality Assurance

- Imagine we have been hired by a candy company to develop a quality assurance software program.
- This software is to ensure that every batch of **jellybeans** delivered to customers has at least **10% of each of the colours** of jellybeans produced by this candy company.
- **Step 1 – Problem Statement**

  Output a report detailing the percentage of each jellybean colour present in every batch of **jellybeans** delivered to customers.

Step 2 - Design

- Machines are good at doing repetitive tasks quickly, so we decide to install a camera at the factory and use a **computer vision** solution to solve the problem.

Get an image

- Download: **http://goo.gl/BXwfZf**

Step 2 - Design

- Machines are good at doing repetitive tasks quickly, so we decide to install a camera at the factory and use a **computer vision** solution to solve the problem.

- **Problem-solving Strategy 1**
  - Divide and conquer:
    - When you have a big problem, divide this big problem into smaller problems and try to solve each of these smaller problems in turn!
      - For example:
        - Let's first start with the **yellow** jellybeans i.e., compute and output the percentage of **yellow** jellybeans

Step 2 - Design

- Write your algorithm

```
# Import image processing libraries

# Define a function colour(r,g,b) to return the colour of a pixel

# Open an image of jellybeans and load its bitmap

# Create a list to store the pixels that are yellow

# Go through all the pixels in the image

# If it's yellow, then add that pixel to the yellow list

# Get the length of the yellow pixel list (same as the number of yellow pixels)

# Calculate the percentage of yellow pixels over the total number of pixels in the image

# Output a report

# Close the image
```

Step 2 - Design

- **Problem-solving Strategy 2**
  - Incremental development and testing

- Start by developing and testing the smallest piece of code you can, then build from there!

- For example:
  - It looks like **image[0,0]** should be **yellow**. Does my function work?

- **Problem-solving Strategy 3**
  - Debug as you go, using `print()` to inspect the value of your variables

What can I develop & test first

Let's step into …

- **Step 3 – Implementation**

```
# Import image processing libraries

# Define a function colour(r,g,b) to return the colour of a pixel

# Open an image of jellybeans and load its bitmap

# Print the value of the topmost pixel -> yellow?

# Test my function to make sure it detects yellow pixel properly

# Close the image
```

- **Step 4 - Testing**

What can I develop & test next

- **Step 3 – Implementation**

```
# Import image processing libraries

# Define a function colour(r,g,b) to return the colour of a pixel

# Open an image of jellybeans and load its bitmap

# Create a list to store the pixels that are yellow

# Go through all the pixels in the image

# If it's yellow, then add that pixel to the yellow list

# Get the length of the yellow pixel list (same as the number of yellow pixels)

# Compute the total number of pixels in the image

# Close the image
```

- **Step 4 - Testing**

What can I develop & test next

- **Step 3 – Implementation**

```
# Import image processing libraries

# Define a function colour(r,g,b) to return the colour of a pixel

# Open an image of jellybeans and load its bitmap

# Create a list to store the pixels that are yellow

# Go through all the pixels in the image

# If it's yellow, then add that pixel to the yellow list

# Get the length of the yellow pixel list (same as the number of yellow pixels)

# Calculate the percentage of yellow pixels over the total number of pixels in the image

# Close the image
```

- **Step 4 - Testing**

What can I develop & test next

- **Step 3 – Implementation**

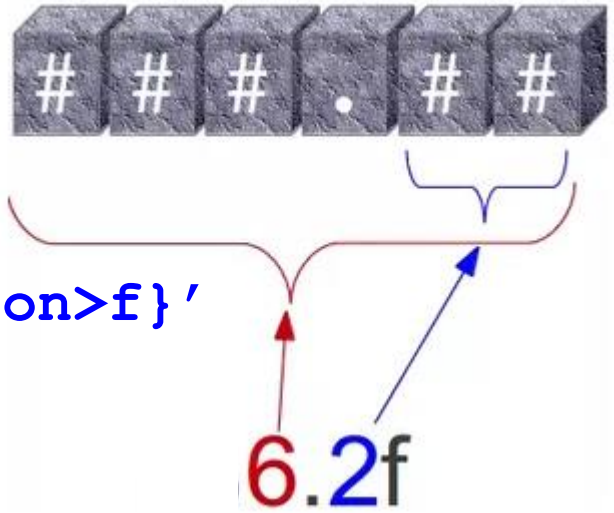
```
# Import image processing libraries

# Define a function colour(r,g,b) to return the colour of a pixel

# Open an image of jellybeans and load its bitmap

# Create a list to store the pixels that are yellow

# Go through all the pixels in the image

# If it's yellow, then add that pixel to the yellow list

# Get the length of the yellow pixel list (same as the number of yellow pixels)

# Calculate the percentage of yellow pixels over the total number of pixels in the image

# Output a report

# Close the image
```

- **Step 4 - Testing**

Printing `float` values

- Set width
- Set precision
- Syntax:

```
f'{<value>:<width>.<precision>f}'
```



Examples:

- `myFloat = 1.25712`
- `myNFloat = f'{myFloat:12.2f}'`
- `print(f'"{myNFloat}"')`
- `"        1.26"`
- `print(f'"{myFloat:12.3f}"')`
- `"       1.257"`

Source: https://python-course.eu/python-tutorial/formatted-output.php

String formatting for output

Examples

Print out an integer

```
totalPixels = 640*480
report = f"There are {totalPixels} pixels."
print(report)
```

```
There are 307200 pixels.
```

Print out a float

```
yellowRatio = 18.122552
report = f"There are {yellowRatio}% yellow pixels."
print(report)
```

```
There are 18.122552% yellow pixels.
```

Print out a float with 3 decimals

```
yellowRatio = 18.122552
report = f"There are {yellowRatio:.3f}% yellow pixels."
print(report)
```

```
There are 18.123% yellow pixels.
```

Print out multiple values with 2 decimals

```
yellowRatio = 18.122552
blueRatio = 10.24521
report = f"We have {yellowRatio:.2f}% yellow and {blueRatio:.2f}% blue."
print(report)
```

```
We have 18.12% yellow and 10.25% blue.
```

Another ways of solving this problem?

That was fun! You try now!

- **Step 1 – Problem Statement**

  Modify the code to check visually that you are capturing the yellow jellybeans.

  Do this by creating an output image where **yellow** pixels are set to **white**.

Review Questions

1. What is one problem solving strategy when writing a big program?
2. How do you format 3.141592654… such that it prints 3.14?

Next lecture

- Start looking ***under the hood*** of the computer and have a look at how the computer memory remembers!