National University of Singapore
School of Computing
CS1010S: Programming Methodology
Semester II, 2024/2025

Recitation 5
Data Abstraction

Release date: 17th February 2025
Due: 23rd February 2025, 23:59

Note that not every problem will be discussed in detail during the recitation. While solutions will be provided after all recitation sessions, you are encouraged to use the Coursemology forums to clarify any doubts that you have. The **Appendix** may be helpful in solving these problems.

General Restrictions

- No importing of additional packages (e.g. `math`). Necessary packages will be stipulated in the template file.
- Use only `tuple` as your compound data structure. No `list`, `set`, `dict`, etc.

Required Files

- r5_images.py
- rec05-template.py

Making an image

This recitation and Mission 3 are centred around common **image processing** operations. We will first describe and implement both the `Pixel` and `Image` Abstract Data Type (ADT) using the `tuple` data structure, then make use of the `Image` ADT to perform image processing!

Pixel ADT

A *pixel* represents the smallest unit of an image, defining its color using the RGB color model. In this abstraction, each pixel is represented by three values: red, green, and blue, which determine its color intensity. Each value ranges from 0 to 255, where 0 represents no intensity, and 255 represents full intensity. The following is the specification of the `Pixel` ADT:

Here, `func_name(args) -> return_val` concisely describes the name, inputs and outputs of a function.

Constructor

- `make_pixel(R, G, B) -> pixel`
  Creates a pixel with the specified red, green, and blue values.

Accessors

- `get_r(pixel) -> int`
  Retrieves the red component of the pixel.
- `get_g(pixel) -> int`
  Retrieves the green component of the pixel.
- `get_b(pixel) -> int`
  Retrieves the blue component of the pixel.

We have defined two special `pixels` as global variables, which you may freely use in your solutions.

```
# Defined inside r5_images.py, which is imported into template file
>>> WHITE = make_pixel(255, 255, 255)
>>> BLACK = make_pixel(0, 0, 0)
```

You are given $0 \leq R, G, B \leq 255$.

A possible implementation of the Pixel ADT using the `tuple` data structure, provided in `rec05-template.py` is as follows:

```
def make_pixel(R, G, B):
    return (R, G, B)

def get_r(pixel):
    return pixel[0]

def get_g(pixel):
    return pixel[1]

def get_b(pixel):
    return pixel[2]
```

Sample execution:

```
>>> pixel = make_pixel(155, 89, 208)
>>> get_r(pixel)
    155
>>> get_g(pixel)
    89
>>> get_b(pixel)
    208
```

Image ADT

An image is a structured collection of *pixels* arranged in a grid. Instead of directly representing the image using raw data structures like tuples, we use an abstraction that provides functions for creating and manipulating images. The following is the specification of the `Image` ADT:

Constructor

- `make_image(width, height) -> image`
  Creates a **white** image grid of the given dimensions.

Accessors

- `get_pixel(image, row, col) -> pixel`
  Retrieves the pixel at a specific `(row, col)` location.
- `get_width(image) -> int`
  Retrieves the width of the image (number of columns).
- `get_height(image) -> int`
  Retrieves the height of the image (number of rows).

Mutators

- `set_pixel(image, row, col, pixel) -> image`
  Updates the pixel at `(row, col)` with the specified color.

Operations

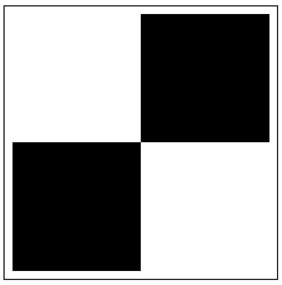
- `display(image) -> None`
  Displays the image (Make sure you do not close the tk window, it will open when you run the program). **This function has been implemented for you.**

Write the functions `make_image`, `get_pixel`, `get_width`, `get_height` and `set_pixel`.

You are given that $0 <$ height, $0 <$ width, $0 \leq$ row $<$ height, $0 \leq$ col $<$ width, i.e. that the inputs are sensible.

Sample execution:

```
# Create an image using the abstraction
>>> checker = make_image(2, 2)  # Create a 2x2 white image
>>> get_width(checker)
    2
>>> get_height(checker)
    2
# Set (0, 1) to black
>>> checker = set_pixel(checker, 0, 1, BLACK)
# Set (1, 0) to black
>>> checker = set_pixel(checker, 1, 0, BLACK)
>>> pixel = get_pixel(checker, 0, 1)
>>> get_r(pixel)
    0
>>> display(checker) # See below
```


(a) `display(checker)`

Loading an `Image` from data

The original inspiration for this question comes from Python Imaging Library (PIL, the package which you installed in Mission 0), where Image.getdata() returns a 1D sequence of pixels after reading in images, which required processing before displaying.

There can be many different ways to represent the pixels that comprise an image. Knowing how to process data to work with the ADTs you create is an essential skill.

We have created two tuples containing the pixels making up two images `felix` and `luna`

```
# Defined inside r5_images.py, which is imported into template file
>>> FELIX = (pixel, pixel, pixel, ...)
>>> LUNA = (pixel, pixel, pixel, ...)
```

For each tuple `IMAGE` containing the pixels of an image with dimensions `width` and `height`, `IMAGE[i * width + j]` gives you the pixel at row `i` and column `j`, where $0 \leq$ `i` $<$ `height` and $0 \leq$ `j` $<$ `width`.

Extending Image ADT, write a function `load_image_from_data` with the following signature:

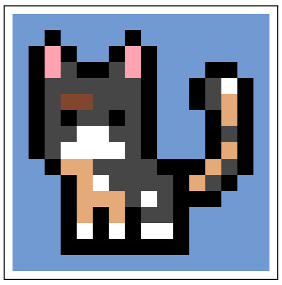
- `load_image_from_data(image, data) -> image`

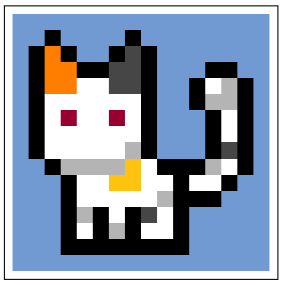
It should take in an `image` ADT instance and a tuple `data` as an input and output a new image as an instance of the Image ADT. Ensure that data abstraction is not broken.

In Coursemology, all functions in the previous question have been implemented for you.

Sample execution:

```
>>> felix_width = felix_height = 16
>>> luna_width = luna_height = 16
>>> felix = make_image(felix_width, felix_height)
>>> felix = load_image_from_data(felix, FELIX)
>>> luna = make_image(luna_width, luna_height)
>>> luna = load_image_from_data(luna, LUNA)
>>> display(felix) # See below
>>> display(luna) # See below
```


(a) `display(felix)`


(b) `display(luna)`

Cropping an `Image`

Cropping is a basic operation in image processing, allowing us to *zoom in* on a region of interest. We can specify the region of interest through the rows and columns that we want to include.

Extending Image ADT, write a function `crop` with the following signature:

- `crop(image, row_from, col_from, row_to, col_to) -> image`
- `crop(image, row_from, col_from, row_to, col_to) -> image`

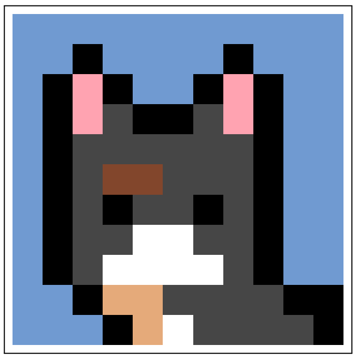
The inputs `row_from` and `row_to` respectively indicate the starting and ending rows of the region of interest, both **inclusive**. The same applies to `col_from` and `col_to`.

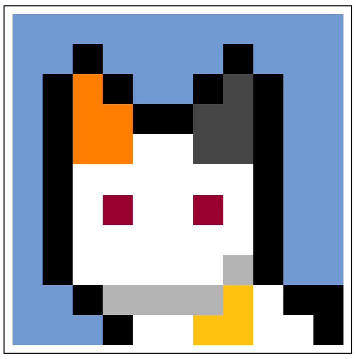
The inputs are always valid, `0` $\leq$ `row_from` $\leq$ `row_to` $<$ `get_height(image)` and `0` $\leq$ `col_from` $\leq$ `col_to` $<$ `get_width(image)`.

In Coursemology, all functions in the previous question have been implemented for you.

Sample execution:

```
>>> felix_head = crop(felix, 0, 0, 10, 10)
>>> get_width(felix_head)
    11
>>> display(felix_head) # See below
>>> display(crop(luna, 0, 0, 10, 10)) # See below
```


(a) `display(felix_head)`


(b) `display(crop(luna, 0, 0, 10, 10))`

Appendix

Abstract Data Type

Abstract Data Types (ADTs) are high-level descriptions of data structures that define the operations possible on the data and the properties of those operations, without specifying implementation details.

`tuple`, an immutable compound data structure

It is easier to just illustrate. A quick summary of common operations.

```
>>> seq = (0, '1', 2, 3, 4) # Create
>>> x = seq[1]              # Index
>>> x in seq                # Check for membership
    True
>>> y = seq[2:len(seq):1]   # Slice
>>> (x,) + y                # Concatenate
    ('1', 2, 3, 4)
>>> a = ()                  # Empty tuple
>>> b = (1,)                # Length 1 tuple
>>> c = (1)                 # Number 1
>>> d = ((),)               # Nested tuple
>>> () + ()
    ()
>>> (1,) + ((),)
    (1, ())
>>> (1,) + (1)
    TypeError
```