Photon

An image processing language.

Akira Higaki
Calum McCartan
Franky Campuzano
Phu Pham

Motivation

- Image editing and processing software is pretty cool.
- Even as technology becomes more advanced, tedious grunt-work is still required, e.g. rotoscoping, greenscreen.
- Automate some of these processes, and provide foundational image processing and manipulation functions with Photon.

Overview of Language

Syntax: C-like syntax

Variable Hoisting: JavaScript like.

Keywords: func, return, if, else, for while

Primitives: int, float, string, bool, pint

Structures: array, Pixel, Image

Colour Alias: _red, _blue, _yellow, _green

Operators: add, subtract pints, Pixels, Images

Built-in: min, max, sqrt, load, save, to_gray, flip, invert, paste

Some of Photon’s extended features

Min, max, sqrt: operations on numeric types, useful for image calculations such as distance, colour aliases arithmetic, etc.

Line comment: hash character

#This is a comment.

String: *string* type and printing strings

Automatic Type Casting

```
int i;
i = 5;

float f;
f = i;
```

- Numeric types are automatically casted to the target type
- This is required for:
  - Binary operations
  - Assignments
  - Function arguments & return statements
- Supported conversions include:
  - Pint -> Int
  - Pint -> Float
  - Int -> Pint
  - Int -> Float
- Supported conversions are enforced by the semantic checker
- Actual conversions done using llvm produced by codegen

Primitive Data Types and Arrays

Primitives: int, float, string, bool, pint

```
int x;
x = 5;
```

```
pint p;
p = 112;
```

```
string word;
word = "hello";
```

Arrays: Non-essential component, provide further functionality to Photon users.

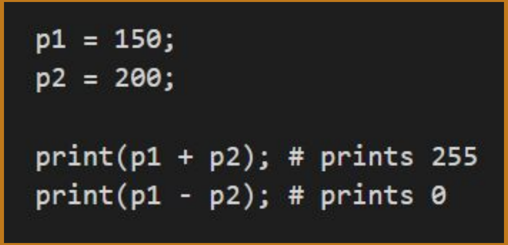
Initialized with **primitive** types, have **dynamic** size. Arrays can be appended, possess length attribute, and have element retrieval.

```
int[] a;
a = [0,2,4];
```

```
array_add(a, 6);
print(a.length); # -> 4
print(a[1]); # -> 2
```

Photon´s types
(pint, pixel, image)

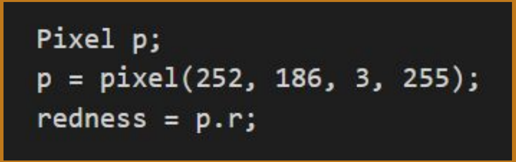
Pint (Pixel-int)



```
p1 = 150;
p2 = 200;

print(p1 + p2); # prints 255
print(p1 - p2); # prints 0
```

- An unsigned 8-bit integer (0-255)
- **Integer overflow is automatically avoided**
- This is done by:
    - Casting both pints to integers
    - Performing the operation
    - Clamping using a pair of select statements
    - Casting the result back to a pint
- Comes with a performance overhead, but very useful for common pixel operations
- 4 pints fit perfectly in the space of a single 32-bit integer...

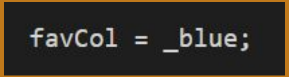
Pixel



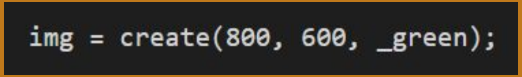
```
Pixel p;
p = pixel(252, 186, 3, 255);
redness = p.r;
```

- A 32-bit struct containing 4 pint values to represent RGBA
- Can be constructed using the pixel() function
- Attributes can be easily accessed
- Pass by value
- Very convenient for using with image functions

Color Aliases

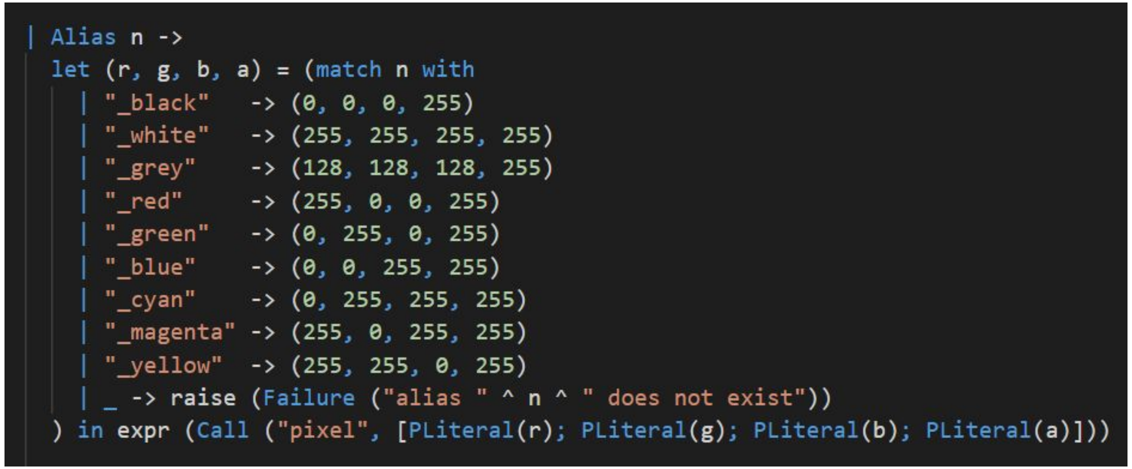


```
favCol = _blue;
```



```
img = create(800, 600, _green);
```

- Short way of calling pixel()
- Substitution performed by the semantic checker
- Convenient with our image functions



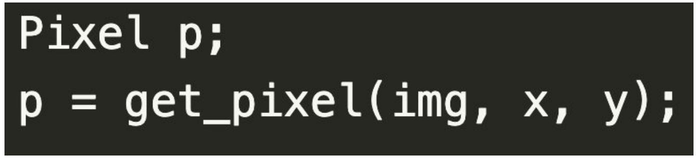
```
| Alias n ->
  let (r, g, b, a) = (match n with
      | "_black"   -> (0, 0, 0, 255)
      | "_white"   -> (255, 255, 255, 255)
      | "_grey"    -> (128, 128, 128, 255)
      | "_red"     -> (255, 0, 0, 255)
      | "_green"   -> (0, 255, 0, 255)
      | "_blue"    -> (0, 0, 255, 255)
      | "_cyan"    -> (0, 255, 255, 255)
      | "_magenta" -> (255, 0, 255, 255)
      | "_yellow"  -> (255, 255, 0, 255)
      | _ -> raise (Failure ("alias " ^ n ^ " does not exist"))
  ) in expr (Call ("pixel", [PLiteral(r); PLiteral(g); PLiteral(b); PLiteral(a)]))
```

Image

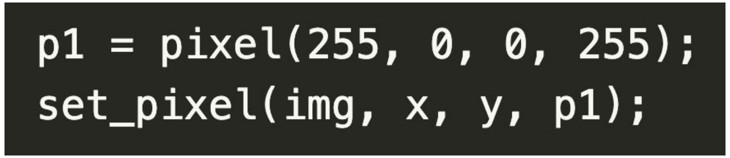
```
Image img;

img = load("Shapes.png");
```

- Images in Photon are implemented using the stb_image C library.
- They are structs containing width, height, size, channels, and data values. Passed by reference.
- At its core, Image in photon are one-dimensional arrays, where every four elements represent RGBA.
- Can be used with the +/- operator to call image_add() and image_subtract().

get_pixel() and set_pixel()



```
Pixel p;
p = get_pixel(img, x, y);
```



```
p1 = pixel(255, 0, 0, 255);
set_pixel(img, x, y, p1);
```

- Returns a pixel containing the red, green, blue, and alpha values at the specified x (width), y (height) coordinates.

- Similarly, setting a pixel requires you pass in the desired pixel along with the coordinates.

A brief overview of some built-in image functions

load(), save(), and destroy()

```
img = load("Shapes.png");
```

For best results, Photon strongly recommends the use of .png images that have a bit depth of 8 or 32.

```
save(img, "ShapesSaved.png");
```

Images must be saved into the same directory as the executable.

```
destroy(img);
```

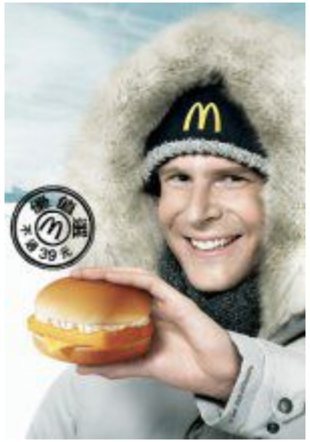
Images are passed by reference, and should be manually destroyed.

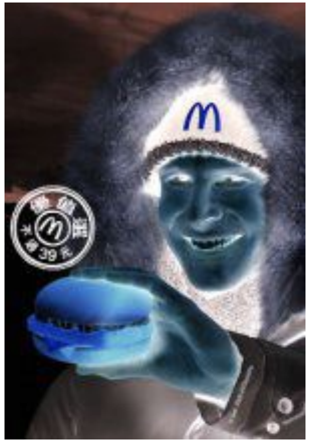
Frees the memory allocated to the image in the heap.

image_invert()

```
newimg = image_invert(imgedwards);
save(newimg, "ImgInvertTest.png");
```

Pass in an image, and image_invert() return an inverted copy.

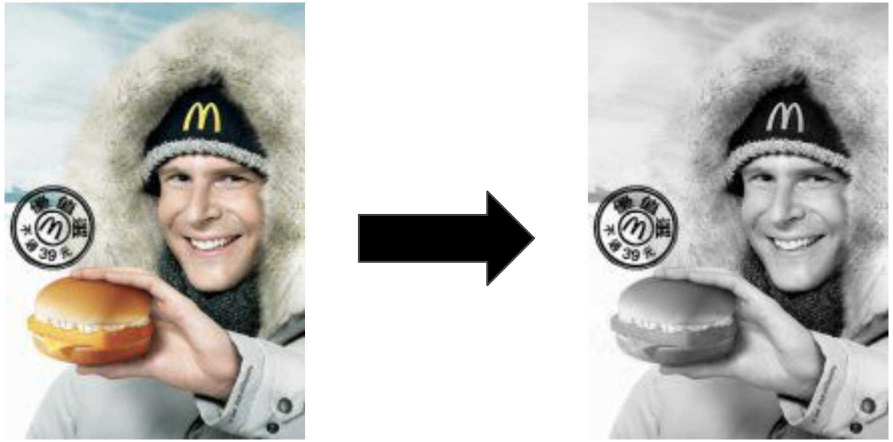




to_gray()

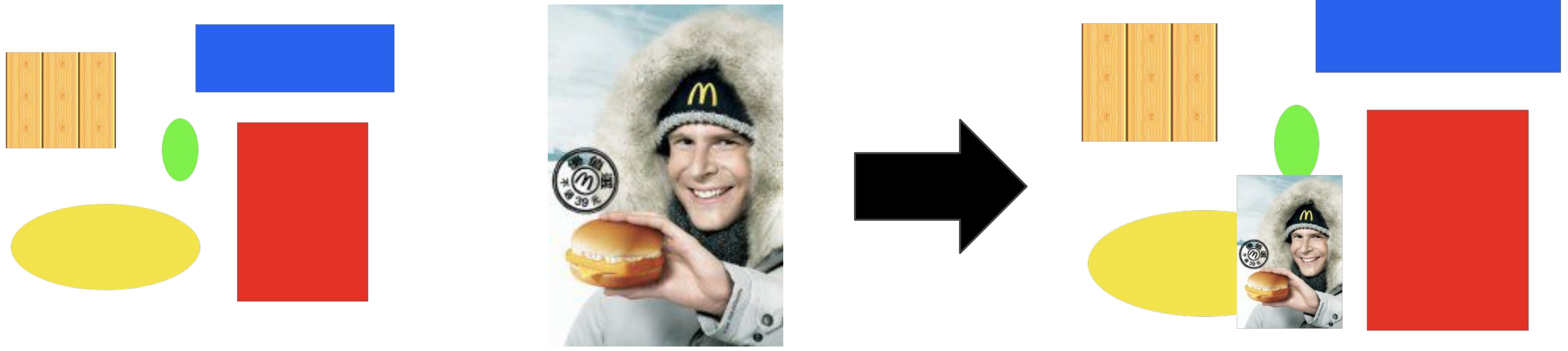
```
grayimg = to_gray(imgedwards);
save(grayimg, "grayEdwards.png");
```

Pass in an image, and to_gray() return a grayscale-ed version.



image_paste()

```
newimg = image_paste(imgshapes, imgedwards, x,y);
save(newimg, "ImgPasteTest.png");
```



Pass in two images, a target and source, along with x, y coordinates.

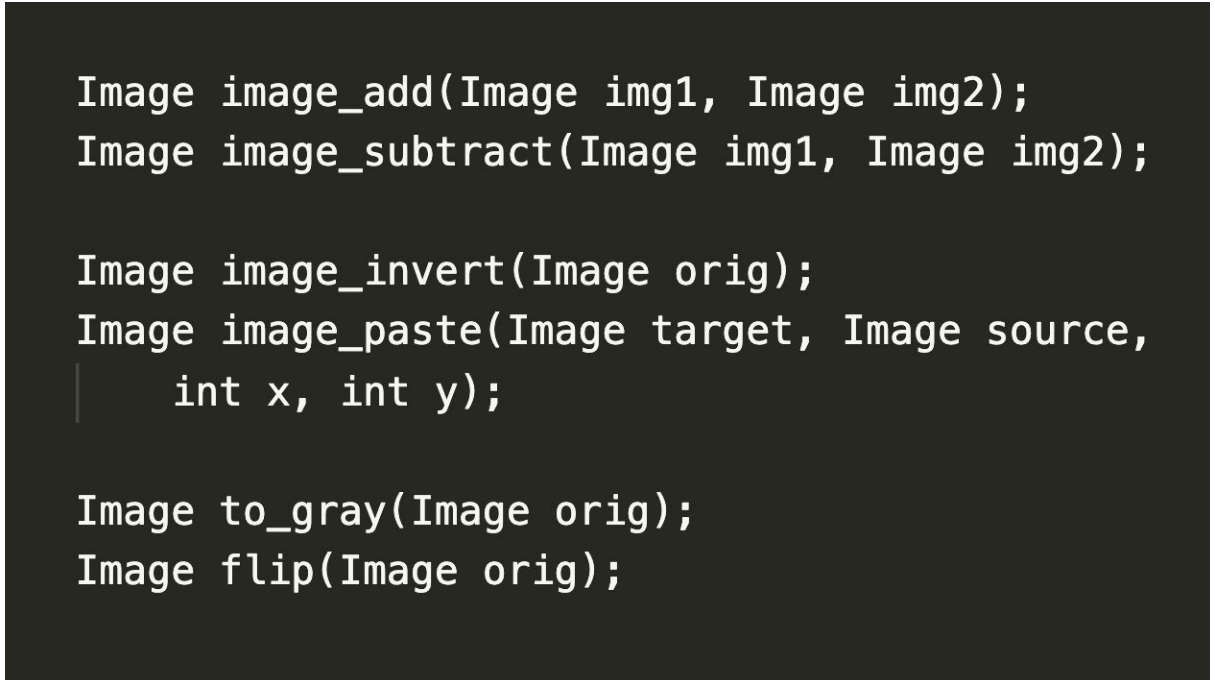
All Built-in Image and Pixel Functions

1/2

```
Image load(string filename);
void save(Image img, string filename);

int width(Image img);
int height(Image img);

Image create( Int width, Int height, Pixel p);
Pixel get_pixel(Image img, int x, int y);
int set_pixel(Image img, int x, int y, Pixel p);
Pixel pixel(pint r, pint g, pint b, pint a);
```

All Built-in Image and Pixel Functions

2/2



```
Image image_add(Image img1, Image img2);
Image image_subtract(Image img1, Image img2);

Image image_invert(Image orig);
Image image_paste(Image target, Image source,
    int x, int y);

Image to_gray(Image orig);
Image flip(Image orig);
```

Future Work:

What Photon *Could* Be

- Increasing the number of compatible image types, even video types
- More built in functions (rotate, move, scale, mask, rotoscoping)
- More merge functions
- Runtime error handling
- Fully implemented arrays and matrices for image representation

Demo Time!

Thanks!