Tutorial 1
Computer vision with python
TA Contact Information

- Peleg harel
- pelegh@post.bgu.ac.il
- Reception hours - Monday 17:00 - 19:00 (after class).
  (Please try to notice what you wish to ask)
What to expect from the practical sessions?

- All things relating to programming.
- A chance to review lecture material via exercises
- I will sometimes ask for your participation
Homework

- You will have 5 HW assignments
- The weight of the homework is 35% or 40%
- Each assignment might include both programming tasks and theoretical questions
- The homework may be done in pairs
- Late submission - Please refer to the Late submission policy page
```python
import this
```

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
Concepts
Duck typing

- If it walks like a duck and it quacks like a duck, then it must be a duck
- Variable has no type - object has types.

```python
In [2]: def foo(a, b):
   print(a + b)

foo(1,2)
foo("a", "b")
```

```
3
ab
```
Functions

- Definition contains function name and parameters name.
- Because of duck typing, names are important!
- No ; - End of statement is end of line.
- And of function body is defined by indentation level.
- Beware of mixing spaces and tabs
Collections

- List - Elements have order.
- Set - A collection of unique elements
- Dictionary - A Mapping from keys to values. Both can have any type
List comprehension

- A clear and readable way to transform lists
- Prefer this over raw for loops
- Can be used for all types of collections
For loops

- Iterates over a collection or an iterator (no c-style loops)
- Can have any number of instructions in loop body
- Same as functions - end of definition is defined by indentation level.
- **When transforming or filtering a list, prefer list comprehension.**

```python
In [15]: for x in range(5):
   ...:     print("Hello")
   ...:     print(x)
```

```
Hello
0
Hello
1
Hello
2
Hello
3
Hello
4
```

```python
In [16]: for x in ['cat', 'mouse', 'dog']:
   ...:     print(x)
```

```
cat
mouse
dog
```
Array slicing

- A single element can be accessed using square brackets (e.g. arr[2] is the third element)
- To access element from the end, use a negative index (e.g. arr[-1] is the last element)
- Use ‘:’ to access a slice of the array. Using this method

```python
latters = ['a', 'b', 'c', 'd']
print(latters[0])  # a
print(latters[-1]) # d
print(latters[1:])  # ['b', 'c', 'd']
print(latters[1:])  # ['b', 'c', 'd']
print(latters[1:2]) # ['b']
print(latters[1:-1]) # ['b', 'c']
```
Python references

https://docs.python.org/3/tutorial/ - official python tutorial
Workspace

- The recommended environment for the course is Jupyter notebook.
- Jupyter notebook is a common tool for scientific programming.
- It includes live coding, interactive widgets, live graph plotting, and more.
- There are two options to work with Jupyter notebooks:

  1. **Google colab** - An online Jupyter notebook environment. Can be accessed without installation.
  2. Local installation - Requires installation of Anaconda and Opencv. Instructions can be found on the course site.
Working with images
Image representation

- A matrix of pixels
- (Row, Column) based indexing - (0, 0) is at the top left corner
- Pixel values are between in a range (min_val, max_val)
- min_val represents black and max_val represents white
- Common pixel ranges are (0 - 1) and (0 - 255)
- (0 - 255) - Is the most common representation used in applications.
- (0 - 1) Will be the preferred representation in the course.
Why 255?

- The smallest unit of memory on a computer is a byte
- A byte consists of 8 bit. Hence it’s range is:

\[2^8 = 256\]
Main workflow

- Read Image from file
- Process the Image
- Display results and annotations

- OpenCV
- Numpy
- SciPy
- Matplotlib
Libraries

- **OpenCV** - The largest open source computer vision library. Written in C++ and has an official python binding.
- **NumPy/SciPy** - Python libraries for linear algebra and other mathematical operations. Operations are implemented in a low level languages and are extremely efficient.
- **Matplotlib** - Python 2D plotting library
- Some operators have both an OpenCV and a NumPy implementation (e.g. convolution)
Read and display an image

- An image will be read as a numpy array
- Use matplotlib to display the image
Numpy arrays

- Multi dimensional extension to python array
- Has a shape and a datatype
- Supports element wise operations
- Supports matrix operations

```python
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])

## Scalar ops - element wise
add_1 = arr1 + 1 #
mult_2 = arr1 * 2

## Array ops - element wise
add_arrays = arr1 + arr2
mult_array = arr1 * arr2 # Note- not matrix multiplication

## Mat ops
arr_transpose = arr1.T
mat_mul = arr1 @ arr2
```
Numpy arrays slicing

- Slicing and element access as similar to python built-in arrays
- Useful to clip images
Pitfall #1 - opencv reads images at uint8 dtype

- Opencv reads images as type unsinged int8 which is limited to range (0, 255)
- To avoid this problem, convert to range (0, 1) and type float

```python
duck_uint8 = cv2.imread('duck.jpg', cv2.IMREAD_GRAYSCALE)
print("uint8 image:")
print(duck_uint8[:5, :5])
print("uint8 times 2 wrong answer:")
print(duck_uint8[:5, :5] * 2)

# instead, convert to float and range (0, 1)
duck_float = (cv2.imread('duck.jpg', cv2.IMREAD_GRAYSCALE).astype(np.float) / 255)

print("float image:")
print(duck_float[:5, :5])
print("float image times 2:")
print(duck_float[:5, :5] * 2)
```
Pitfall #2 - matplotlib vmin and vmax

- The default behavior of imshow is to deduce min and max values from the given arrays. Can be problematic for uniform arrays.
- Use vmin and vmax to set them manually.
Pitfall #2 - matplotlib vmin and vmax

```
[110] img_zeros = np.zeros((5, 5))
    img_ones = np.ones((5, 5))

plt.imshow(img_zeros, cmap='gray')
plt.imshow(img_ones, cmap='gray')
```

```
plt.imshow(img_zeros,
    cmap='gray',
    vmin=0,
    vmax=1)
```

```
plt.imshow(img_ones,
    cmap='gray',
    vmin=0,
    vmax=1)
```
Pitfall #3 - (row, column) vs (x, y)

- Array access in numpy is row, column based
- Other functions such as image processing of opencv or matplotlib uses (x, y) to represent image locations
- To deal with it, use meaningful variable names (e.g. row, col for indices and x,y for points)