



CSC 3141

IMAGE PROCESSING LABORATORY

01 – Introduction to Python

Course Details

- **Evaluation :**

Assignments/ Quizzes : 20% - 10%

Individual project : 30%

Final Examination : 50% - 60%

- **Recommended text:**

- *Gonzalez, R. and Woods, R. (2008). Digital Image Processing, 3rd Ed., Prentice Hall.*
- *OpenCV documentation, <https://docs.opencv.org/master/>*
- *Solem, J.E. (2012). Programming Computer Vision with Python: Tools and algorithms for analyzing images, 1st Ed. O'Reilly.*

Terminology

An Image:

An image is defined as a two-dimensional function, $F(x,y)$, where x and y are spatial coordinates, and the amplitude of F at any pair of coordinates (x,y) is called the **intensity** of that image at that point.

Digital Image:

When (x,y) and amplitude values of F are finite, we call it a digital image.

Digital Processing:

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it.

Types of Image Processing

Analog Image Processing:

The analog image processing is applied on analog signals and it processes only two-dimensional signals. The images are manipulated by electrical signals. In analog image processing, analog signals can be periodic or non-periodic

Digital Image Processing:

A digital image processing is applied to digital images (a matrix of small pixels and elements). For manipulating the images, there is a number of software and algorithms that are applied to perform changes.



Python Basics

Introduction to Python

- Python is an interpreted, object oriented, high-level, general-purpose programming language.
- Created by Guido van Rossum in 1991
- Python is a programming language that lets you work quickly and integrate systems more efficiently.
- Loosely typed (no need to declare the type)
- Indentation based code block separation (for statement grouping)
- Open Source! Free!



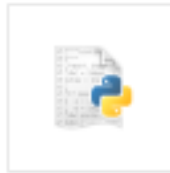
Installing and setting up Python

- Download Python with the following link
["https://www.python.org/downloads/"](https://www.python.org/downloads/) (For references)
- Choose the one appropriate for your OS and architecture(32-bit or 64-bit).



IDLE

- IDLE is an Integrated Development Environment for Python.
- IDLE has two main windows type, the Shell window (interactive interpreter) and the Editor window.

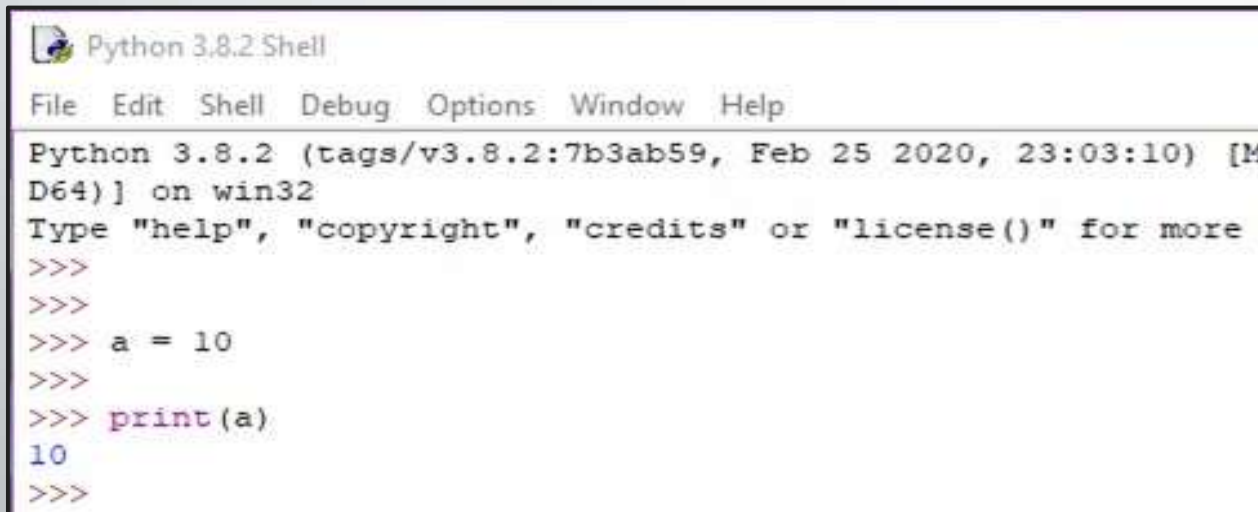


IDLE (Python 3.8 64-bit)

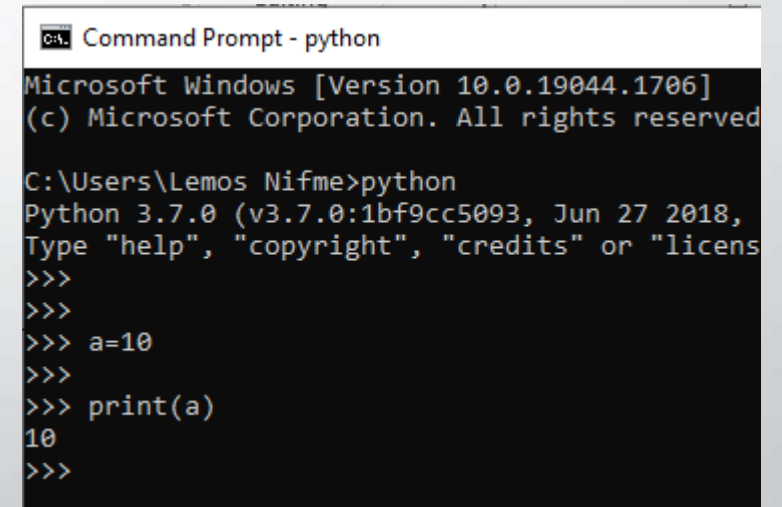
App

Python Shell

- Python shell which is used to execute a single python command and get the result. (*Two variations: IDLE (GUI), Python [command line]*)



```
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [AMD64] on win32
Type "help", "copyright", "credits" or "license()" for more
>>>
>>>
>>> a = 10
>>>
>>> print(a)
10
>>>
```



```
Command Prompt - python
Microsoft Windows [Version 10.0.19044.1706]
(c) Microsoft Corporation. All rights reserved

C:\Users\Lemos Nifme>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018,
Type "help", "copyright", "credits" or "license()" for more
>>>
>>>
>>> a=10
>>>
>>> print(a)
10
>>>
```

print() and input()

print() : Produces text output on the console.

Syntax:

```
print("<Message>")           OR   print('<Message>')  
Ex: print('Hello World')      >>> Hello World
```

Input() : Reads a value from user input (always reads as a string).

Syntax:

```
<variable> = input("<Message>")  
Ex: a = input("Enter a number:")    >>> Enter a number: 6
```

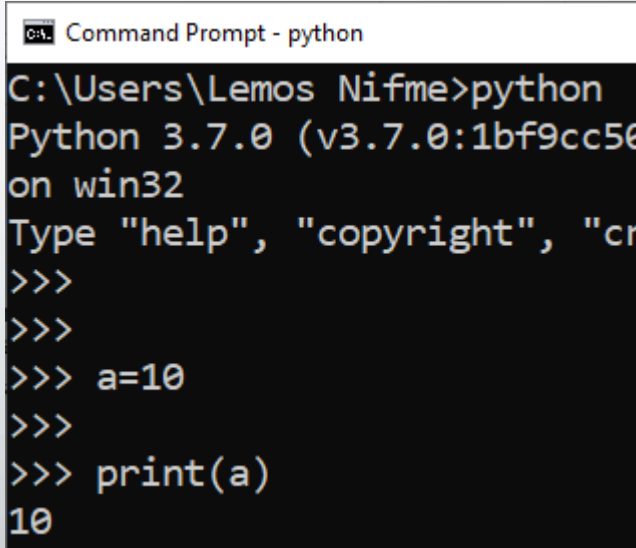
Variables

- **Variable:** A named piece of memory that can store a value.
A variable consists of an identifier and a memory address
- **Assignment statement:** Stores a value into a variable

Syntax:

<name> = <value>

- An identifier must obey the following rules.
 - ✓ Must start with a letter or an underscore.
 - ✓ Can contain letters, digits, and underscores.
 - ✓ Can be of any length.
 - ✓ Cannot be a Python keyword.



```
Command Prompt - python
C:\Users\Lemos Nifme>python
Python 3.7.0 (v3.7.0:1bf9cc50
on win32
Type "help", "copyright", "cr
>>>
>>>
>>> a=10
>>>
>>> print(a)
10
```

Functions

Syntax:

```
def function_name( parameters ):
    """function_docstring"""
    function_suite
    return [expression]
```

Function Header

Function Body

Functions Examples

def keyword function name function parameters

def add(x, y): Colon marks the end of the function header

function code {
 print("Sum of number", x, "and ", y)
 sum = x + y
 return sum
}

function return statement

```
def basic_operations(a, b):  
    total = a + b  
    dif = a - b  
    product = a * b  
    div = a / b  
  
    return total, dif, product, div
```

Data types

- *Booleans* are either True or False.
- *Numbers* can be integers (1 and 2), floats (1.1 and 1.2), fractions (1/2 and 2/3), or even complex numbers.
- *Strings* are sequences of Unicode characters, e.g. an html document.
- *Bytes* and byte arrays, e.g. a jpeg image file.
- ***Lists*** are ordered sequences of values.
- ***Tuples*** are ordered, immutable sequences of values.
- ***Sets*** are unordered bags of values.
- ***Dictionaries*** are unordered bags of key-value pairs.

Numeric

- int - Integer number (28, 19, -7)
- float - Real number (0.556, -2.65)
- complex - Complex number (5 + 2j)
- bool - Boolean (True, False)
- **Typecasting numbers:**
 - *"float()" , "int()"*

Operators

Arithmetic operators:

Operators	Meaning	Example	Result
+	Addition	4 + 2	6
-	Subtraction	4 - 2	2
*	Multiplication	4 * 2	8
/	Division	4 / 2	2
%	Modulus operator to get remainder in integer division	5 % 2	1
**	Exponent	5**2 = 5 ²	25
//	Integer Division/ Floor Division	5//2 -5//2	2 -3

Logical operators:

Operator	Meaning	Example	Result
and	Logical and	(5<2) and (5>3)	False
or	Logical or	(5<2) or (5>3)	True
not	Logical not	not (5<2)	True

Assignment operators:

Operator	Example	Equivalent Expression (m=15)	Result
=	y = a+b	y = 10 + 20	30
+=	m +=10	m = m+10	25
-=	m -=10	m = m-10	5
*=	m *=10	m = m*10	150
/=	m /=10	m = m/10	1.5
%=	m %=10	m = m%10	5
=	m=2	m = m**2 or $m = m^2$	225
//=	m//=10	m = m//10	1

Relational / comparison operators:

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
!=	Not equal to	5!=2	True

Strings

- Python has a built-in string class named "str" with many handy features.
- String literals can be enclosed by either double or single quotes.
- Python strings can be indexed and sliced in exactly the same way as lists.

Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	strA + strB will give Hello World!
*	Repetition - Creates new strings, concatenating multiple copies of the same string	strA * 2 will give -Hello Hello
[]	Slice - Gives the character from the given index	strA[1] will give e
[:]	Range Slice - Gives the characters from the given range	strA[1:4] will give ell
in	Membership - Returns true if a character exists in the given string	H in strA will give 1
not in	Membership - Returns true if a character does not exist in the given string	M not in strA will give 1
r/R	Raw String - Suppresses actual meaning of Escape characters.	print r'\n' prints \n and print R'\n' prints \n
%	Format - Performs String formatting	It's the same like we used in C.

Strings Contd.

String concatenation

- The “+” operator is used to concatenate two strings.
- The “*str()*” method is used to convert values to a string.

String operations

- `capitalize()` - Capitalizes first letter of string
- `upper()` - Converts lowercase letters in string to uppercase.
- `lower()` - Converts all uppercase letters in string to lowercase.
- `swapcase()` - Inverts case for all letters in string.
- `isupper()` - Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.



Python

Collection Data Types

`[]`, `()`, `{}`, `{k:v}`

Lists

- A collection of values
- Contain any type
- Contain different types
- 0-based indexing

Syntax: [`<e1>`,`<e12>`,...]

```
a = []      # a is an empty list
b = ['python', 1, False]

print(a, len(a))
print(b, len(b))
```

List Indexing

- List items can be accessed using the “*list[i]*” statement where “*i*” is the index and “*list*” is the identifier of the list.
- A negative index accesses a list from the end counting backwards.

Ex:

```
list_1 = ["Kandy","Colombo","Galle","Jaffna"," Ampara"]
```

```
print( list_1[1] )
```

```
>>>"Colombo"
```

```
print( list_1[-1] )
```

```
>>>"Ampara"
```

```
print( list_1[1:3] )
```

```
>>>["Colombo","Galle"]
```

List slicing

- Obtaining a part of a list once it has been defined is called *slicing* a list.
- A part of a list, called a “slice”, is obtained by specifying two indices. This returns a new list containing all the items of the original list, in order, starting from the first slice index, up to but not including the second slice index.

`<list>[<>:<length_of_the_index>]`

Ex:

`list_1[:3]` returns 0,1,2 indexed elements from list_1

Adding items to a list

- The “+” operator concatenates two lists to make a new one.

```
a = ['red', 'green']  
b = [1, 2]  
  
a = a + b    # The "+" operator creates a new list and assigns it to a.  
print(a)
```

- The “*append()*” method adds a single item to the end of a list

```
a = ['red', 'green']  
  
a.append('blue')  
print(a)
```

Adding items to a list

- The "*extend()*" method takes one list or a single value as an argument and appends each item of this list to the original list.

```
a = ['red', 'green']
a.extend([1, 2, 3])    # Adds all the elements in the argument list
                       # to original list.
print(a)

b = ['cat', 'dog']
a.extend(b)           # This works the same.
print(a)
```


Adding items to a list (continued)

- The `insert()` method inserts a single item to the list at the given index. The first argument is the index to be inserted at, and the second argument is the item to be inserted.

```
a = ['red', 'green', 'blue']  
a.insert(1, 'black')    # Inserts the string "black" at index 1.  
  
# The value "green" which was at index 1 gets bumped to index 2, and  
# "black" is inserted at index 1.  
print(a)
```

Tuple

- Immutable/unchangeable
(cannot add or remove items after tuple is created)
- Use full for fixed data
- Faster than list lists (accessing items and iteration)

Syntax: <identifier> = (<el1>, <el2>,)

Ex:

- `tuple1 = (1,2,3,4,5)`
- `tuple2 = ('a', 'b', 'c')`
- `tuple3 = ('a', 3, 45.9, 'Hello')`

Set

- Store non duplicate items
- unindexed
- Very fast access compared to Lists
- Math set operations(union, intersect)
- Unordered

Syntax: <identifier> = {<el1>, <el2>,}

Dictionaries

- A dictionary , or a “dict”, is an efficient key/value hash table structure Python.
- The contents of a dictionary is a series of ordered key/value pairs.
from python version 3.7, dictionaries are ordered
- Dictionaries are changeable (can change, add or remove items after creation)
- Dictionaries do not allow duplicates (do not allow duplicated keys)

Dictionaries (Cont...)

Syntax:

```
<dict name> = {<key>:<value>}
```

- dict is listed in curly brackets.
- Inside the curly brackets, keys and values are declared.
- Each key is separated from its value by a colon(:) while each element is separated by commas.

Dictionaries (Contd.)

- **Key –**
 - No duplicate key is allowed.
 - Immutable data type such as string, numbers, or tuples.
 - Case sensitive.

```
>>> new_dict1 = {1:"First year", 2:"Second Year", 3:"Third Year", 4:"Fourth Year"}  
>>>  
>>> print(new_dict1)  
{1: 'First year', 2: 'Second Year', 3: 'Third Year', 4: 'Fourth Year'}
```

Dictionaries (Cont...)

```
>>> dict_course = {"CS100":553, "CS104":349, "CS202":294, "CS314":13}
>>> dict_student = {"First year":('Praveen','Ruwandi','Salith'),"Second year":('Anushka','Chanaka','Himansi'),"Third Year-SP":('Gunasekara','Edirisinghe','Madugalla')}
>>>
```

```
>>> dict_student["First year"]
('Praveen', 'Ruwandi', 'Salith')
>>>
>>>
>>> dict_student["First Year"]
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    dict_student["First Year"]
KeyError: 'First Year'
>>> |
```

Dictionary in-built methods

- `keys()` - Retrieve the list of keys in the dictionary.
- `values()` - Retrieve the list of values in the dictionary.
- `items()` - Retrieve the list of keys and their values as tuples in the dictionary.
- `copy()` - Copy the entire dictionary to new dictionary.
- `update()` - Update a dictionary by adding a new entry or a key:value pair to an existing entry.



Python

Repetition and Selection

The For loop

- **for loop:** Repeats a set of statements over a group of values.

Syntax:

```
for variable_name in collection:  
    statements
```

- If the index is required, we can use the “*enumerate()*” function.

```
numbers = [1, 2, 3, 4, 5]  
total = 0  
  
for number in numbers:  
    total += number  
  
print(total)
```

```
numbers = [10,20,30,40,50]  
total = 0  
  
for index,number in enumerate(numbers):  
    total += number  
    print(index)  
  
print("\n",total)
```

Range

- The **range** function specifies a range of integers:
 - `range(start, stop)` - the integers between **start** (inclusive) and **stop** (exclusive)
 - `range(start, stop, step)` - the integers between **start** (inclusive) and **stop** (exclusive) by **step**

Example :

```
for x in range(5, 0, -1):  
    print (x)
```

If, If-else, else-if statements

1. **if** --- Syntax:

```
if condition:  
    #statements
```

The statements are executed if condition is evaluated to True.

2. **if-else** --- Syntax:

```
if condition:  
    #statements  
else:  
    #statements
```

Executes one block of statements if a certain condition is True, and a second block of statements if it is False

3. **else-if** --- Syntax:

```
if condition:  
    #statements  
elif condition:  
    #statements  
else:  
    #statements
```

Multiple conditions can be chained with **elif** ("else if")

While-loop

- Python also has the standard while-loop.
- The while loop provides total control over the index numbers.

Syntax:

```
while condition:  
    #statements
```

```
a = [0, 1, 2, 3, 4, 5, 6, 7, 8]  
i = 0  
  
while i < len(a):  
    print(a[i])  
    i += 2
```



Python

File Handling

File Handling

- The key function : **open()**

Syntax : <file_object> = open(<filename> , <mode>)

- modes:
 - “r” - Read mode which is used when the file is only being read
 - “w” - Write mode which is used to edit and write new information to the file (any existing files with the same name will be erased when this mode is activated)
 - “a” - Appending mode, which is used to add new data to the end of the file; that is new information is automatically amended to the end
 - “r+” - Special read and write mode, which is used to handle both actions when working with a file
 - “b” - Open in binary mode

Example:

(Constant file name)

```
inputFile = open("data.txt", "r")
```

or

(Variable file name: entered by user at runtime)

```
fname = input("Enter name of input file: ")
```

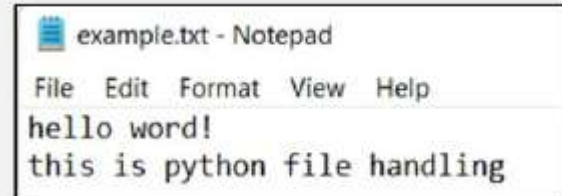
```
inputFile = open(fname, "w")
```


Read file

- “r” – opens a file for reading
- read() – Reading the content of the file
- read([size]) – Reading the content of the file for a specified size
- readline() – Read one line of the file
- readlines() – Read lines to a list

```
f= open("example.txt","r+")  
  
print( f.read(10) ) # size=10 → 10 bytes/ 10  
                    characters  
  
f.close()
```

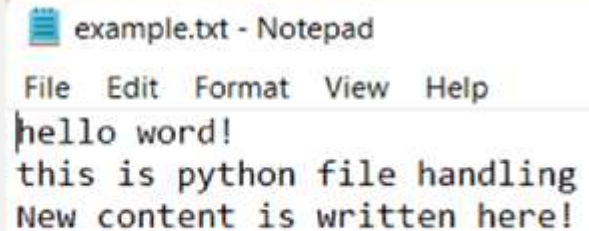
#output
hello word



Write file

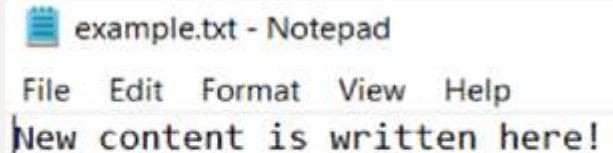
- “w” – Write : will overwrite any existing content
- “a” – Append : will append to the end of the file

```
f= open("example.txt","a") #append  
f.write("New content is written here!")  
f.close()
```

A screenshot of a Notepad window titled "example.txt - Notepad". The menu bar includes "File", "Edit", "Format", "View", and "Help". The text content of the file is "hello word!", "this is python file handling", and "New content is written here!".

example.txt - Notepad
File Edit Format View Help
hello word!
this is python file handling
New content is written here!

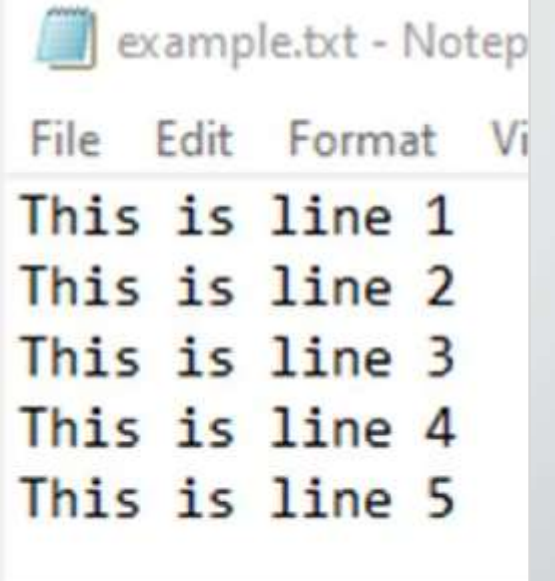
```
f= open("example.txt","w") #overwrite  
f.write("New content is written here!")  
f.close()
```

A screenshot of a Notepad window titled "example.txt - Notepad". The menu bar includes "File", "Edit", "Format", "View", and "Help". The text content of the file is "New content is written here!".

example.txt - Notepad
File Edit Format View Help
New content is written here!

Writing multiple lines

```
f= open("example.txt","w")  
  
for i in range(5):  
    f.write("This is line %d\n" % (i+1))  
  
f.close()
```



Note: You should always close your files at the end. In some cases, due to buffering, changes made to a file may not be visible until you close the file.

Delete file & Rename file

Syntax (Delete):

```
import os  
os.remove(<file name>)
```

Syntax (Rename):

```
import os  
os.rename(<file name>, <new file name>)
```

– END –

