CODING BOYS(REVA UNIVERSITY)

PROJECT NAME- IMAGE PROCESSING USING PYTHON

TEAM MEMBERS:

NAVEEN SRIPAD VEERAPUR

PAVAN JOGALE

OMKAR BHAGAT

PRANJAL SINGH

SIMPLEITK BASIC CONCEPTS

In [35]:

```
!pip install SimpleITK
```

```
Requirement already satisfied: SimpleITK in c:\users\91866\anaconda3\lib\site-packages (2.0.1)
```

In [34]:

```
from __future__ import print_function
%matplotlib inline
import matplotlib.pyplot as plt
import SimpleITK as sitk
```

Image Construction

There are a variety of ways to create an image. All images' initial value is well defined as zero.

In [11]:

```
image = sitk.Image(256, 128, 64, sitk.sitkInt16)
image_2D = sitk.Image(64, 64, sitk.sitkFloat32)
image_2D = sitk.Image([32,32], sitk.sitkUInt32)
image_RGB = sitk.Image([128,128], sitk.sitkVectorUInt8, 3)
```

Pixel Types

The pixel type is represented as an enumerated type. The following is a table of the enumerated list.

sitkUInt8	Unsigned 8 bit integer
sitkInt8	Signed 8 bit integer
sitkUInt16	Unsigned 16 bit integer
sitkInt16	Signed 16 bit integer
sitkUInt32	Unsigned 32 bit integer
sitkInt32	Signed 32 bit integer
sitkUInt64	Unsigned 64 bit integer
sitkInt64	Signed 64 bit integer
sitkFloat32	32 bit float
sitkFloat64	64 bit float
sitkComplexFloat32	complex number of 32 bit float
sitkComplexFloat64	complex number of 64 bit float

sitkComplexFloat64	complex number of 64 bit float
sitkVectorUInt8	Multi-component of unsigned 8 bit integer
sitkVectorInt8	Multi-component of signed 8 bit integer
sitkVectorUInt16	Multi-component of unsigned 16 bit integer
sitkVectorInt16	Multi-component of signed 16 bit integer
sitkVectorUInt32	Multi-component of unsigned 32 bit integer
sitkVectorInt32	Multi-component of signed 32 bit integer
sitkVectorUInt64	Multi-component of unsigned 64 bit integer
sitkVectorInt64	Multi-component of signed 64 bit integer
sitkVectorFloat32	Multi-component of 32 bit float
sitkVectorFloat64	Multi-component of 64 bit float
sitkLabelUInt8	RLE label of unsigned 8 bit integers
sitkLabelUInt16	RLE label of unsigned 16 bit integers
sitkLabelUInt32	RLE label of unsigned 32 bit integers
sitkLabelUInt64	RLE label of unsigned 64 bit integers

There is also `sitkUnknown`, which is used for undefined or erroneous pixel ID's. It has a value of -1.

The 64-bit integer types are not available on all distributions. When not available the value is `sitkUnknown`.

More Information about the Image class be obtained in the Docstring

SimpleITK classes and functions have the Docstrings derived from the C++ definitions and the Doxygen documentation.

In [12]:

```
help(image)
```

```
Help on Image in module SimpleITK.SimpleITK object:

class Image(builtins.object)
 |  Image(*args)
 |
 |  Proxy of C++ itk::simple::Image class.
 |
 |  Methods defined here:
 |
 |  CopyInformation(self, srcImage)
 |      CopyInformation(Image self, Image srcImage)
 |
 |  EraseMetaData(self, key)
 |      EraseMetaData(Image self, std::string const & key) -> bool
 |
 |  GetDepth(self)
 |      GetDepth(Image self) -> unsigned int
 |
 |  GetDimension(self)
 |      GetDimension(Image self) -> unsigned int
 |
 |  GetDirection(self)
 |      GetDirection(Image self) -> VectorDouble
 |
 |  GetHeight(self)
 |      GetHeight(Image self) -> unsigned int
 |
 |  GetITKBase(self, *args)
 |      GetITKBase(Image self) -> itk::DataObject
 |      GetITKBase(Image self) -> itk::DataObject const *
 |
 |  GetMetaData(self, key)
 |      GetMetaData(Image self, std::string const & key) -> std::string
 |
 |  GetMetaDataKeys(self)
 |      GetMetaDataKeys(Image self) -> VectorString
 |
 |  GetNumberOfComponentsPerPixel(self)
```

```
 |  GetNumberOfComponentsPerPixel(self)
 |      GetNumberOfComponentsPerPixel(Image self) -> unsigned int
 |
 |  GetNumberOfPixels(self)
 |      GetNumberOfPixels(Image self) -> uint64_t
 |
 |  GetOrigin(self)
 |      GetOrigin(Image self) -> VectorDouble
 |
 |  GetPixel(self, *idx)
 |      Returns the value of a pixel.
 |
 |      This method takes 2 parameters in 2D: the x and y index,
 |      and 3 parameters in 3D: the x, y and z index.
 |
 |  GetPixelAsComplexFloat64(self, idx)
 |      GetPixelAsComplexFloat64(Image self, VectorUInt32 idx) -> std::complex< double >
 |
 |  GetPixelID(self)
 |      GetPixelID(Image self) -> itk::simple::PixelIDValueEnum
 |
 |  GetPixelIDTypeAsString(self)
 |      GetPixelIDTypeAsString(Image self) -> std::string
 |
 |  GetPixelIDValue(self)
 |      GetPixelIDValue(Image self) -> itk::simple::PixelIDValueType
 |
 |  GetSize(self)
 |      GetSize(Image self) -> VectorUInt32
 |
 |  GetSpacing(self)
 |      GetSpacing(Image self) -> VectorDouble
 |
 |  GetWidth(self)
 |      GetWidth(Image self) -> unsigned int
 |
 |  HasMetaDataKey(self, key)
 |      HasMetaDataKey(Image self, std::string const & key) -> bool
 |
 |  IsUnique(self)
 |      IsUnique(Image self) -> bool
 |
 |  MakeUnique(self)
 |      MakeUnique(Image self)
 |
 |  SetDirection(self, direction)
 |      SetDirection(Image self, VectorDouble direction)
 |
 |  SetMetaData(self, key, value)
 |      SetMetaData(Image self, std::string const & key, std::string const & value)
 |
 |  SetOrigin(self, origin)
 |      SetOrigin(Image self, VectorDouble origin)
 |
 |  SetPixel(self, *args)
 |      Sets the value of a pixel.
 |
 |      This method takes 3 parameters in 2D: the x and y index then the value,
 |      and 4 parameters in 3D: the x, y and z index then the value.
 |
 |  SetPixelAsComplexFloat64(self, idx, v)
 |      SetPixelAsComplexFloat64(Image self, VectorUInt32 idx, std::complex< double > const v)
 |
 |  SetSpacing(self, spacing)
 |      SetSpacing(Image self, VectorDouble spacing)
 |
 |  TransformContinuousIndexToPhysicalPoint(self, index)
 |      TransformContinuousIndexToPhysicalPoint(Image self, VectorDouble index) -> VectorDouble
 |
 |  TransformIndexToPhysicalPoint(self, index)
 |      TransformIndexToPhysicalPoint(Image self, VectorInt64 index) -> VectorDouble
 |
 |  TransformPhysicalPointToContinuousIndex(self, point)
 |      TransformPhysicalPointToContinuousIndex(Image self, VectorDouble point) -> VectorDouble
 |
 |  TransformPhysicalPointToIndex(self, point)
 |      TransformPhysicalPointToIndex(Image self, VectorDouble point) -> VectorInt64
```

```
 |
 |  __GetPixelAsComplexFloat32__(self, idx)
 |      __GetPixelAsComplexFloat32__(Image self, VectorUInt32 idx) -> std::complex< float >
 |
 |  __GetPixelAsDouble__(self, idx)
 |      __GetPixelAsDouble__(Image self, VectorUInt32 idx) -> double
 |
 |  __GetPixelAsFloat__(self, idx)
 |      __GetPixelAsFloat__(Image self, VectorUInt32 idx) -> float
 |
 |  __GetPixelAsInt16__(self, idx)
 |      __GetPixelAsInt16__(Image self, VectorUInt32 idx) -> int16_t
 |
 |  __GetPixelAsInt32__(self, idx)
 |      __GetPixelAsInt32__(Image self, VectorUInt32 idx) -> int32_t
 |
 |  __GetPixelAsInt64__(self, idx)
 |      __GetPixelAsInt64__(Image self, VectorUInt32 idx) -> int64_t
 |
 |  __GetPixelAsInt8__(self, idx)
 |      __GetPixelAsInt8__(Image self, VectorUInt32 idx) -> int8_t
 |
 |  __GetPixelAsUInt16__(self, idx)
 |      __GetPixelAsUInt16__(Image self, VectorUInt32 idx) -> uint16_t
 |
 |  __GetPixelAsUInt32__(self, idx)
 |      __GetPixelAsUInt32__(Image self, VectorUInt32 idx) -> uint32_t
 |
 |  __GetPixelAsUInt64__(self, idx)
 |      __GetPixelAsUInt64__(Image self, VectorUInt32 idx) -> uint64_t
 |
 |  __GetPixelAsUInt8__(self, idx)
 |      __GetPixelAsUInt8__(Image self, VectorUInt32 idx) -> uint8_t
 |
 |  __GetPixelAsVectorFloat32__(self, idx)
 |      __GetPixelAsVectorFloat32__(Image self, VectorUInt32 idx) -> VectorFloat
 |
 |  __GetPixelAsVectorFloat64__(self, idx)
 |      __GetPixelAsVectorFloat64__(Image self, VectorUInt32 idx) -> VectorDouble
 |
 |  __GetPixelAsVectorInt16__(self, idx)
 |      __GetPixelAsVectorInt16__(Image self, VectorUInt32 idx) -> VectorInt16
 |
 |  __GetPixelAsVectorInt32__(self, idx)
 |      __GetPixelAsVectorInt32__(Image self, VectorUInt32 idx) -> VectorInt32
 |
 |  __GetPixelAsVectorInt64__(self, idx)
 |      __GetPixelAsVectorInt64__(Image self, VectorUInt32 idx) -> VectorInt64
 |
 |  __GetPixelAsVectorInt8__(self, idx)
 |      __GetPixelAsVectorInt8__(Image self, VectorUInt32 idx) -> VectorInt8
 |
 |  __GetPixelAsVectorUInt16__(self, idx)
 |      __GetPixelAsVectorUInt16__(Image self, VectorUInt32 idx) -> VectorUInt16
 |
 |  __GetPixelAsVectorUInt32__(self, idx)
 |      __GetPixelAsVectorUInt32__(Image self, VectorUInt32 idx) -> VectorUInt32
 |
 |  __GetPixelAsVectorUInt64__(self, idx)
 |      __GetPixelAsVectorUInt64__(Image self, VectorUInt32 idx) -> VectorUInt64
 |
 |  __GetPixelAsVectorUInt8__(self, idx)
 |      __GetPixelAsVectorUInt8__(Image self, VectorUInt32 idx) -> VectorUInt8
 |
 |  __SetPixelAsComplexFloat32__(self, idx, v)
 |      __SetPixelAsComplexFloat32__(Image self, VectorUInt32 idx, std::complex< float > const v)
 |
 |  __SetPixelAsDouble__(self, idx, v)
 |      __SetPixelAsDouble__(Image self, VectorUInt32 idx, double v)
 |
 |  __SetPixelAsFloat__(self, idx, v)
 |      __SetPixelAsFloat__(Image self, VectorUInt32 idx, float v)
 |
 |  __SetPixelAsInt16__(self, idx, v)
 |      __SetPixelAsInt16__(Image self, VectorUInt32 idx, int16_t v)
 |
 |  __SetPixelAsInt32__(self, idx, v)
```

```
 |      __SetPixelAsInt32__(Image self, VectorUInt32 idx, int32_t v)
 |
 |  __SetPixelAsInt64__(self, idx, v)
 |      __SetPixelAsInt64__(Image self, VectorUInt32 idx, int64_t v)
 |
 |  __SetPixelAsInt8__(self, idx, v)
 |      __SetPixelAsInt8__(Image self, VectorUInt32 idx, int8_t v)
 |
 |  __SetPixelAsUInt16__(self, idx, v)
 |      __SetPixelAsUInt16__(Image self, VectorUInt32 idx, uint16_t v)
 |
 |  __SetPixelAsUInt32__(self, idx, v)
 |      __SetPixelAsUInt32__(Image self, VectorUInt32 idx, uint32_t v)
 |
 |  __SetPixelAsUInt64__(self, idx, v)
 |      __SetPixelAsUInt64__(Image self, VectorUInt32 idx, uint64_t v)
 |
 |  __SetPixelAsUInt8__(self, idx, v)
 |      __SetPixelAsUInt8__(Image self, VectorUInt32 idx, uint8_t v)
 |
 |  __SetPixelAsVectorFloat32__(self, idx, v)
 |      __SetPixelAsVectorFloat32__(Image self, VectorUInt32 idx, VectorFloat v)
 |
 |  __SetPixelAsVectorFloat64__(self, idx, v)
 |      __SetPixelAsVectorFloat64__(Image self, VectorUInt32 idx, VectorDouble v)
 |
 |  __SetPixelAsVectorInt16__(self, idx, v)
 |      __SetPixelAsVectorInt16__(Image self, VectorUInt32 idx, VectorInt16 v)
 |
 |  __SetPixelAsVectorInt32__(self, idx, v)
 |      __SetPixelAsVectorInt32__(Image self, VectorUInt32 idx, VectorInt32 v)
 |
 |  __SetPixelAsVectorInt64__(self, idx, v)
 |      __SetPixelAsVectorInt64__(Image self, VectorUInt32 idx, VectorInt64 v)
 |
 |  __SetPixelAsVectorInt8__(self, idx, v)
 |      __SetPixelAsVectorInt8__(Image self, VectorUInt32 idx, VectorInt8 v)
 |
 |  __SetPixelAsVectorUInt16__(self, idx, v)
 |      __SetPixelAsVectorUInt16__(Image self, VectorUInt32 idx, VectorUInt16 v)
 |
 |  __SetPixelAsVectorUInt32__(self, idx, v)
 |      __SetPixelAsVectorUInt32__(Image self, VectorUInt32 idx, VectorUInt32 v)
 |
 |  __SetPixelAsVectorUInt64__(self, idx, v)
 |      __SetPixelAsVectorUInt64__(Image self, VectorUInt32 idx, VectorUInt64 v)
 |
 |  __SetPixelAsVectorUInt8__(self, idx, v)
 |      __SetPixelAsVectorUInt8__(Image self, VectorUInt32 idx, VectorUInt8 v)
 |
 |  __abs__(self)
 |
 |  __add__(self, other)
 |
 |  __and__(self, other)
 |
 |  __copy__(self)
 |      Create a SimpleITK shallow copy, where the internal image share is shared with copy on
write implementation.
 |
 |  __deepcopy__(self, memo)
 |      Create a new copy of the data and image class.
 |
 |  __div__(self, other)
 |
 |  __eq__(self, other)
 |      Return self==value.
 |
 |  __floordiv__(self, other)
 |
 |  __ge__(self, other)
 |      Return self>=value.
 |
 |  __getitem__(self, idx)
 |      Get an pixel value or a sliced image.
 |
 |      This operator implements basic indexing where idx is
```

```
 |      arguments or a squence of integers the same dimension as
 |      the image. The result will be a pixel value from that
 |      index.
 |
 |      Multi-dimension extended slice based indexing is also
 |      implemented. The return is a copy of a new image. The
 |      standard sliced based indices are supported including
 |      negative indices, to indicate location relative to the
 |      end, along with negative step sized to indicate reversing
 |      of direction.
 |
 |      If the length of idx is less than the number of dimension
 |      of the image it will be padded with the defaults slice
 |      ":".
 |
 |      When an index element is an integer, that dimension is
 |      collapsed extracting an image with reduced dimensionality.
 |      The minimum dimension of an image which can be extracted
 |      is 2D.
 |
 |  __gt__(self, other)
 |      Return self>value.
 |
 |  __iadd__(self, *args)
 |      __iadd__(Image self, Image i) -> Image
 |      __iadd__(Image self, double c) -> Image
 |
 |  __iand__(self, *args)
 |      __iand__(Image self, Image i) -> Image
 |      __iand__(Image self, int c) -> Image
 |
 |  __ifloordiv__(self, *args)
 |      __ifloordiv__(Image self, Image i) -> Image
 |      __ifloordiv__(Image self, double c) -> Image
 |
 |  __imod__(self, *args)
 |      __imod__(Image self, Image i) -> Image
 |      __imod__(Image self, int c) -> Image
 |
 |  __imul__(self, *args)
 |      __imul__(Image self, Image i) -> Image
 |      __imul__(Image self, double c) -> Image
 |
 |  __init__(self, *args)
 |      __init__(Image self) -> Image
 |      __init__(Image self, Image img) -> Image
 |      __init__(Image self, unsigned int width, unsigned int height,
itk::simple::PixelIDValueEnum valueEnum) -> Image
 |      __init__(Image self, unsigned int width, unsigned int height, unsigned int depth,
itk::simple::PixelIDValueEnum valueEnum) -> Image
 |      __init__(Image self, VectorUInt32 size, itk::simple::PixelIDValueEnum valueEnum, unsigned
int numberOfComponents=0) -> Image
 |
 |  __invert__(self)
 |
 |  __ior__(self, *args)
 |      __ior__(Image self, Image i) -> Image
 |      __ior__(Image self, int c) -> Image
 |
 |  __ipow__(self, *args)
 |      __ipow__(Image self, Image i) -> Image
 |      __ipow__(Image self, double c) -> Image
 |
 |  __isub__(self, *args)
 |      __isub__(Image self, Image i) -> Image
 |      __isub__(Image self, double c) -> Image
 |
 |  __iter__(self)
 |
 |  __itruediv__(self, *args)
 |      __itruediv__(Image self, Image i) -> Image
 |      __itruediv__(Image self, double c) -> Image
 |
 |  __ixor__(self, *args)
 |      __ixor__(Image self, Image i) -> Image
 |      __ixor__(Image self, int c) -> Image
 |
```

```
 |  __le__(self, other)
 |      Return self<=value.
 |
 |  __len__(self)
 |
 |  __lt__(self, other)
 |      Return self<value.
 |
 |  __mod__(self, other)
 |
 |  __mul__(self, other)
 |
 |  __ne__(self, other)
 |      Return self!=value.
 |
 |  __neg__(self)
 |
 |  __or__(self, other)
 |
 |  __pos__(self)
 |
 |  __pow__(self, other)
 |
 |  __radd__(self, other)
 |
 |  __rand__(self, other)
 |
 |  __rdiv__(self, other)
 |
 |  __reduce_ex__(self, protocol)
 |      Helper for pickle.
 |
 |  __repr__ = _swig_repr(self)
 |
 |  __rfloordiv__(self, other)
 |
 |  __rmul__(self, other)
 |
 |  __ror__(self, other)
 |
 |  __rpow__(self, other)
 |
 |  __rsub__(self, other)
 |
 |  __rtruediv__(self, other)
 |
 |  __rxor__(self, other)
 |
 |  __setitem__(self, idx, rvalue)
 |      Sets this image's pixel value(s) to rvalue.
 |
 |      The dimension of idx must match that of the image.
 |
 |      If all indices are integers then rvalue should be a pixel value
 |      ( scalar or sequence for vector pixels). The value is assigned to
 |      the pixel.
 |
 |      If the indices are slices or integers then, the PasteImageFilter is
 |      used to assign values to this image. The rvalue can be an image
 |      or a scalar constant value. When rvalue is an image it must be of
 |      the same pixel type and equal or lesser dimension than self. The
 |      region defined by idx and rvalue's size must be compatible. The
 |      region defined by idx will collapse one sized idx dimensions when it
 |      does not match the rvalue image's size.
 |
 |  __setstate__(self, args)
 |
 |  __str__(self)
 |      __str__(Image self) -> std::string
 |
 |  __sub__(self, other)
 |
 |  __truediv__(self, other)
 |
 |  __xor__(self, other)
 |
 |  ----------------------------------------------------------------------
 |  ----------------------------------------------------------------------
```

```
 |  Static methods defined here:
 |  
 |  __swig_destroy__ = delete_Image(...)
 |      delete_Image(Image self)
 |  
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |  
 |  __dict__
 |      dictionary for instance variables (if defined)
 |  
 |  __weakref__
 |      list of weak references to the object (if defined)
 |  
 |  thisown
 |      The membership flag
 |  
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |  
 |  __hash__ = None
```

Accessing Attributes

If you are familiar with ITK, then these methods will follow your expectations:

In [13]:

```
print(image.GetSize())
print(image.GetOrigin())
print(image.GetSpacing())
print(image.GetDirection())
print(image.GetNumberOfComponentsPerPixel())
```

```
(256, 128, 64)
(0.0, 0.0, 0.0)
(1.0, 1.0, 1.0)
(1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0)
1
```

Note: The starting index of a SimpleITK Image is always 0. If the output of an ITK filter has non-zero starting index, then the index will be set to 0, and the origin adjusted accordingly.

The size of the image's dimensions have explicit accessors:

In [14]:

```
print(image.GetWidth())
print(image.GetHeight())
print(image.GetDepth())
```

```
256
128
64
```

Since the dimension and pixel type of a SimpleITK image is determined at run-time accessors are needed.

In [15]:

```
print(image.GetDimension())
print(image.GetPixelIDValue())
print(image.GetPixelIDTypeAsString())
```

```
3
2
16-bit signed integer
```

What is the depth of a 2D image?

In [16]:

```
print(image_2D.GetSize())
print(image_2D.GetDepth())
```

```
(32, 32)
0
```

What is the dimension and size of a Vector image?

In [17]:

```
print(image_RGB.GetDimension())
print(image_RGB.GetSize())
```

```
2
(128, 128)
```

In [18]:

```
print(image_RGB.GetNumberOfComponentsPerPixel())
```

```
3
```

For certain file types such as DICOM, additional information about the image is contained in the meta-data dictionary.

In [19]:

```
for key in image.GetMetaDataKeys():
        print("\"{0}\":\"{1}\"".format(key, image.GetMetaData(key)))
```

Accessing Pixels

There are the member functions `GetPixel` and `SetPixel` which provides an ITK-like interface for pixel access.

In [20]:

```
help(image.GetPixel)
```

```
Help on method GetPixel in module SimpleITK.SimpleITK:

GetPixel(*idx) method of SimpleITK.SimpleITK.Image instance
    Returns the value of a pixel.

    This method takes 2 parameters in 2D: the x and y index,
    and 3 parameters in 3D: the x, y and z index.
```

In [21]:

```
print(image.GetPixel(0, 0, 0))
image.SetPixel(0, 0, 0, 1)
print(image.GetPixel(0, 0, 0))
```

```
0
1
```

In [22]:

```
print(image[0,0,0])
image[0,0,0] = 10
print(image[0,0,0])
```

```
print(image[0,0,0])
```

```
1
10
```

Conversion between numpy and SimpleITK

In [23]:

```
nda = sitk.GetArrayFromImage(image)
print(nda)
```

```
[[[10   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]
  ...
  [ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]]

 [[ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]
  ...
  [ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]]

 [[ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]
  ...
  [ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]]

 ...

 [[ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]
  ...
  [ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]]

 [[ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]
  ...
  [ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]]

 [[ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]
  ...
  [ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]
  [ 0   0   0 ...   0   0   0]]]
```

In [24]:

```
help(sitk.GetArrayFromImage)
```

```
Help on function GetArrayFromImage in module SimpleITK.extra:

GetArrayFromImage(image)
    Get a NumPy ndarray from a SimpleITK Image.

    This is a deep copy of the image buffer and is completely safe and without potential side
```

effects.

In [25]:

```
# Get a view of the image data as a numpy array, useful for display
nda = sitk.GetArrayViewFromImage(image)
```

In [26]:

```
nda = sitk.GetArrayFromImage(image_RGB)
img = sitk.GetImageFromArray(nda)
img.GetSize()
```

Out[26]:

```
(3, 128, 128)
```

In [27]:

```
help(sitk.GetImageFromArray)
```

```
Help on function GetImageFromArray in module SimpleITK.extra:

GetImageFromArray(arr, isVector=None)
    Get a SimpleITK Image from a numpy array.

    If isVector is True, then the Image will have a Vector pixel type, and the last dimension of t
he array will be
    considered the component index. By default when isVector is None, 4D arrays
    are automatically considered 3D vector images, but 3D arrays are 3D images.
```

In [28]:

```
img = sitk.GetImageFromArray(nda, isVector=True)
print(img)
```

```
VectorImage (000001C544430C10)
  RTTI typeinfo:   class itk::VectorImage<unsigned char,2>
  Reference Count: 1
  Modified Time: 690
  Debug: Off
  Object Name:
  Observers:
    none
  Source: (none)
  Source output name: (none)
  Release Data: Off
  Data Released: False
  Global Release Data: Off
  PipelineMTime: 0
  UpdateMTime: 0
  RealTimeStamp: 0 seconds
  LargestPossibleRegion:
    Dimension: 2
    Index: [0, 0]
    Size: [128, 128]
  BufferedRegion:
    Dimension: 2
    Index: [0, 0]
    Size: [128, 128]
  RequestedRegion:
    Dimension: 2
    Index: [0, 0]
    Size: [128, 128]
  Spacing: [1, 1]
  Origin: [0, 0]
  Direction:
1 0
0 1
```

```
  IndexToPointMatrix:
1 0
0 1

  PointToIndexMatrix:
1 0
0 1

  Inverse Direction:
1 0
0 1

  VectorLength: 3
  PixelContainer:
    ImportImageContainer (000001C5452EDF70)
      RTTI typeinfo:   class itk::ImportImageContainer<unsigned __int64,unsigned char>
      Reference Count: 1
      Modified Time: 691
      Debug: Off
      Object Name:
      Observers:
        none
      Pointer: 000001C544316360
      Container manages memory: true
      Size: 49152
      Capacity: 49152
```

The order of index and dimensions need careful attention during conversion

ITK's Image class does not have a bracket operator. It has a GetPixel which takes an ITK Index object as an argument, which is ordered as `(x,y,z)`. This is the convention that SimpleITK's Image class uses for the GetPixel method and slicing operator as well. In numpy, an array is indexed in the **opposite** order `(z,y,x)`. Also note that the access to channels is different. In SimpleITK you do not access the channel directly, rather the pixel value representing all channels for the specific pixel is returned and you then access the channel for that pixel. In the numpy array you are accessing the channel directly.

In [29]:

```
import numpy as np

multi_channel_3Dimage = sitk.Image([2,4,8], sitk.sitkVectorFloat32, 5)
x = multi_channel_3Dimage.GetWidth() - 1
y = multi_channel_3Dimage.GetHeight() - 1
z = multi_channel_3Dimage.GetDepth() - 1
multi_channel_3Dimage[x,y,z] = np.random.random(multi_channel_3Dimage.GetNumberOfComponentsPerPixel
())

nda = sitk.GetArrayFromImage(multi_channel_3Dimage)

print("Image size: " + str(multi_channel_3Dimage.GetSize()))
print("Numpy array size: " + str(nda.shape))

# Notice the index order and channel access are different:
print("First channel value in image: " + str(multi_channel_3Dimage[x,y,z][0]))
print("First channel value in numpy array: " + str(nda[z,y,x,0]))
```

```
Image size: (2, 4, 8)
Numpy array size: (8, 4, 2, 5)
First channel value in image: 0.5717631578445435
First channel value in numpy array: 0.57176316
```

Are we still dealing with Image, because I haven't seen one yet...

While SimpleITK does not do visualization, it does contain a built in `Show` method. This function writes the image out to disk and than launches a program for visualization. By default it is configured to use ImageJ, because it is readily supports all the image types which SimpleITK has and load very quickly. However, it's easily customizable by setting environment variables.

In [ ]:
In [ ]:

```
sitk.Show?
```

By converting into a numpy array, matplotlib can be used for visualization for integration into the scientific python environment.

In [31]:

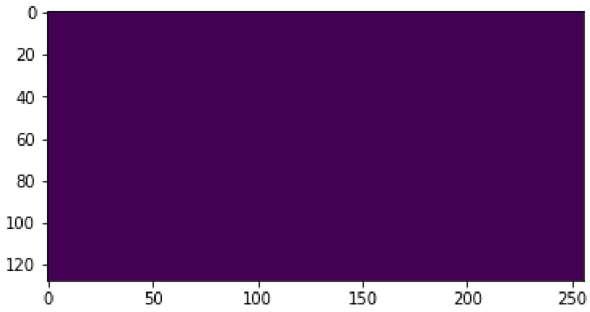
```
%matplotlib inline
import matplotlib.pyplot as plt
```

In [32]:

```
z = 0
slice = sitk.GetArrayViewFromImage(image)[z,:,:]
plt.imshow(slice)
```

Out[32]:

```
<matplotlib.image.AxesImage at 0x1c5475d5e48>
```



Pythonic Syntactic Sugar

launch binder

The Image Basics Notebook was straight forward and closely follows ITK's C++ interface.

Sugar is great it gives your energy to get things done faster! SimpleITK has applied a generous about of syntactic sugar to help get things done faster too.

In [2]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rc('image', aspect='equal')
import SimpleITK as sitk
# Download data to work on
%run update_path_to_download_script
from downloaddata import fetch_data as fdata
```

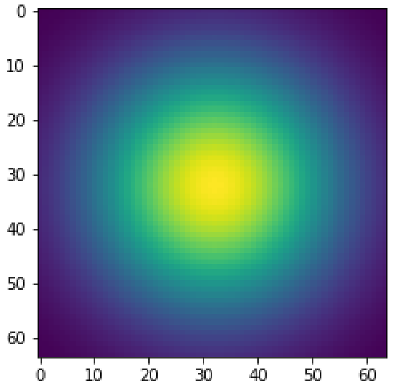
Let us begin by developing a convenient method for displaying images in our notebooks.

In [3]:

```
img = sitk.GaussianSource(size=[64]*2)
plt.imshow(sitk.GetArrayViewFromImage(img))
```

Out[3]:

```
<matplotlib.image.AxesImage at 0x1b0b6361bc8>
```

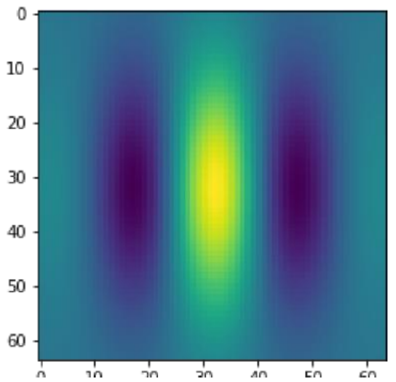


In [4]:

```
img = sitk.GaborSource(size=[64]*2, frequency=.03)
plt.imshow(sitk.GetArrayViewFromImage(img))
```

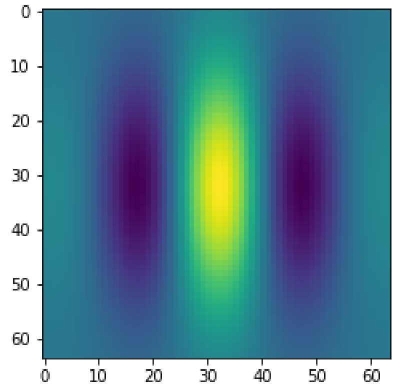
Out[4]:

```
<matplotlib.image.AxesImage at 0x1b0b6418908>
```



In [5]:

```
def myshow(img):
    nda = sitk.GetArrayViewFromImage(img)
    plt.imshow(nda)
myshow(img)
```



Multi-dimension slice indexing

If you are familiar with numpy, sliced index then this should be cake for the SimpleITK image. The Python standard slice interface for 1-D object:

Operation	Result
d[i]	i-th item of d, starting index 0
d[i:j]	slice of d from i to j
d[i:j:k]	slice of d from i to j with step k

With this convenient syntax many basic tasks can be easily done.

In [6]:

```
img[24,24]
```

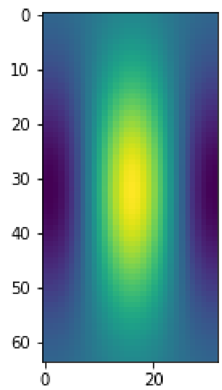
Out[6]:

```
0.048901304602622986
```

Cropping

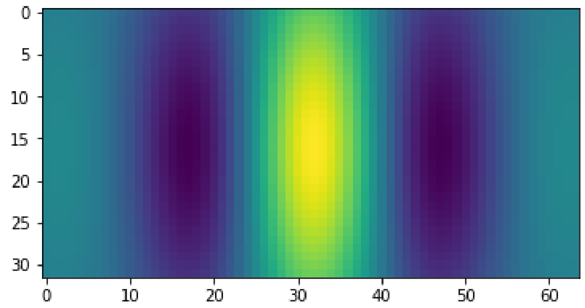
In [7]:

```
myshow(img[16:48,:])
```



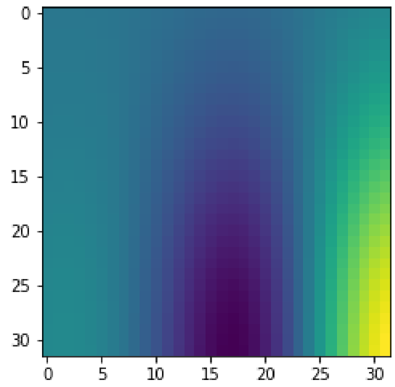
In [8]:

```
myshow(img[:,16:-16])
```



In [9]:

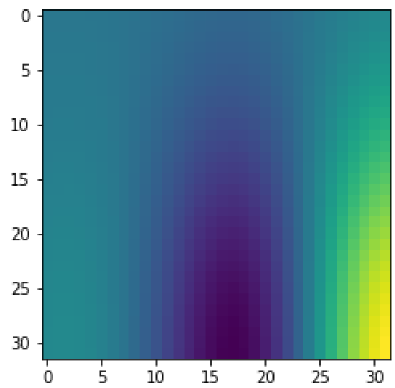
```
myshow(img[:32,:32])
```



Flipping

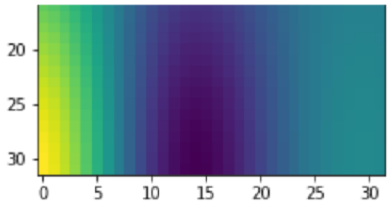
In [10]:

```
img_corner = img[:32,:32]
myshow(img_corner)
```



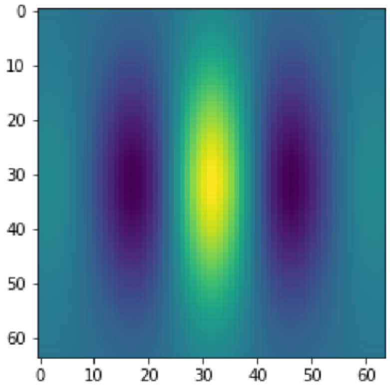
In [11]:

```
myshow(img_corner[::-1,:])
```



In [12]:

```
myshow(sitk.Tile(img_corner, img_corner[::-1,::],img_corner[::,::-1],img_corner[::-1,::-1], [2,2]))
```



Slice Extraction

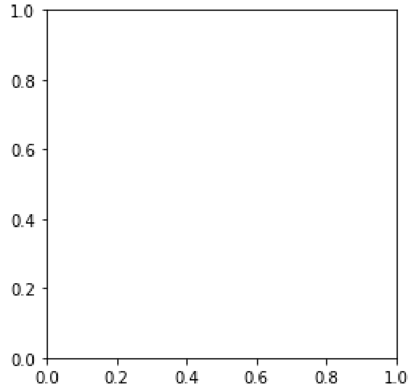
A 2D image can be extracted from a 3D one.

In [13]:

```
img = sitk.GaborSource(size=[64]*3, frequency=0.05)

# Why does this produce an error?
myshow(img)
```

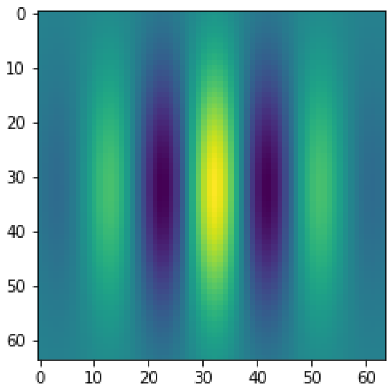
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-e30b71f2ae99> in <module>
      2 
      3 # Why does this produce an error?
----> 4 myshow(img)

<ipython-input-5-66f5146c4fb2> in myshow(img)
      1 def myshow(img):
      2     nda = sitk.GetArrayViewFromImage(img)
----> 3     plt.imshow(nda)
      4 myshow(img)

~\Anaconda3\lib\site-packages\matplotlib\pyplot.py in imshow(X, cmap, norm, aspect, interpolation, alpha, vmin, vmax, origin, extent, shape, filternorm, filterrad, imlim, resample, url, data, **kwargs)
   2681         filternorm=filternorm, filterrad=filterrad, imlim=imlim,
   2682         resample=resample, url=url, **({"data": data} if data is not
-> 2683         None else {}), **kwargs)
   2684     sci(__ret)
   2685     return __ret

~\Anaconda3\lib\site-packages\matplotlib\__init__.py in inner(ax, data, *args, **kwargs)
   1599     def inner(ax, *args, data=None, **kwargs):
   1600         if data is None:
-> 1601             return func(ax, *map(sanitize_sequence, args), **kwargs)
   1602 
   1603         bound = new_sig.bind(ax, *args, **kwargs)

~\Anaconda3\lib\site-packages\matplotlib\cbook\deprecation.py in wrapper(*args, **kwargs)
    367                 f"%(removal)s.  If any parameter follows {name!r}, they "
    368                 f"should be pass as keyword, not positionally.")
--> 369         return func(*args, **kwargs)
```

```
        return wrapper

~\Anaconda3\lib\site-packages\matplotlib\cbook\deprecation.py in wrapper(*args, **kwargs)
                    f"%(removal)s.  If any parameter follows {name!r}, they "
                    f"should be pass as keyword, not positionally.")
-->         return func(*args, **kwargs)

        return wrapper

~\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py in imshow(self, X, cmap, norm, aspect, inter
polation, alpha, vmin, vmax, origin, extent, shape, filternorm, filterrad, imlim, resample, url, *
*kwargs)
                                  resample=resample, **kwargs)

->          im.set_data(X)
            im.set_alpha(alpha)
            if im.get_clip_path() is None:

~\Anaconda3\lib\site-packages\matplotlib\image.py in set_data(self, A)
                    or self._A.ndim == 3 and self._A.shape[-1] in [3, 4]):
                raise TypeError("Invalid shape {} for image data"
-->                             .format(self._A.shape))

            if self._A.ndim == 3:

TypeError: Invalid shape (64, 64, 64) for image data
```



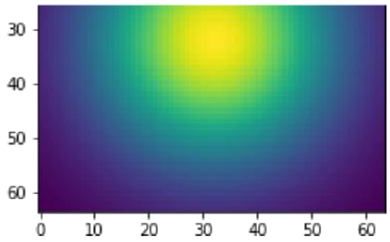
In [14]:

```
myshow(img[:,:,32])
```



In [15]:

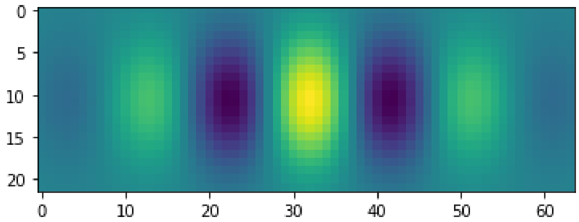
```
myshow(img[16,:,:])
```



Subsampling

In [16]:

```
myshow(img[:,::3,32])
```



Mathematical Operators

Most python mathematical operators are overloaded to call the SimpleITK filter which does that same operation on a per-pixel basis. They can operate on a two images or an image and a scalar.

If two images are used then both must have the same pixel type. The output image type is usually the same.

As these operators basically call ITK filter, which just use raw C++ operators, care must be taken to prevent overflow, and divide by zero etc.

Operators
+
-
*
/
//
**

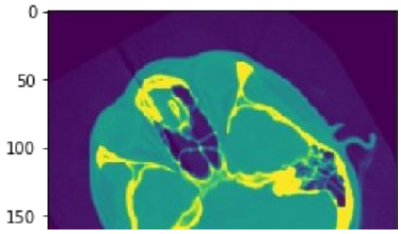
In [17]:

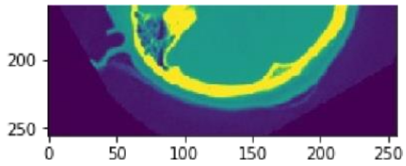
```
img = sitk.ReadImage(fdata("cthead1.png"))
img = sitk.Cast(img,sitk.sitkFloat32)
myshow(img)
img[150,150]
```

```
Fetching cthead1.png
Downloaded 29351 of 29351 bytes (100.00%)
```

Out[17]:

```
140.0
```



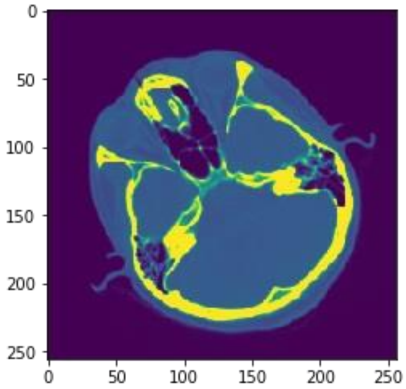


In [18]:

```
timg = img**2
myshow(timg)
timg[150,150]
```

Out[18]:

```
19600.0
```



Division Operators

All three Python division operators are implemented `__floordiv__`, `__truediv__`, and `__div__`.

The true division's output is a double pixel type.

See PEP 238 to see why Python changed the division operator in Python 3.

Bitwise Logic Operators

Operators
&
|
^
~

In [ ]:

```
img = sitk.ReadImage(fdata("cthead1.png"))
myshow(img)
```

Comparative Operators

Operators
>
>=
<
<=
==

These comparative operators follow the same convention as the reset of SimpleITK for binary images. They have the pixel type of

These comparative operators follow the same convention as the reset of SimpleITK for binary images. They have the pixel type of

`sitkUInt8` with values of 0 and 1.

In [ ]:

```
img = sitk.ReadImage(fdata("cthead1.png"))
myshow(img)
```

Amazingly make common trivial tasks really trivial

In [ ]:

```
myshow(img>90)
```

In [ ]:

```
myshow(img>150)
```

In [ ]:

```
myshow((img>90)+(img>150))
```

SYNOPSIS OF IMAGE PROCESSING:

Digital image processing is the use of a digital computer to process digital images through an algorithm. As a subcategory or field of digital signal processing, digital image processing has many advantages over analog image processing. It allows a much wider range of algorithms to be applied to the input data and can avoid problems such as the build-up of noise and distortion during processing. Since images are defined over two dimensions (perhaps more) digital image processing may be modelled in the form of multidimensional systems. The generation and development of digital image processing are mainly affected by three factors: first, the development of computers; second, the development of mathematics (especially the creation and improvement of discrete mathematics theory); third, the demand for a wide range of applications in environment, agriculture, military, industry and medical science has increased.

During doing this project we came to understand the importance of image processing in a developer's life. We are not yet perfect in this field we are still learning to make ourselves better day to day. During this project we experienced a lot of difficulties but finally we worked hard and completed it. We enjoyed this project a lot and we as a team thank u for this wonderful opportunity. We have also decided to do the same project in our university mini project.

Please, follow the below given link for a video description about the topic by one of our team member omkar bhagat. So please do visit the link once:

https://drive.google.com/file/d/14ueojA91pFBd9vXDwlIf2ufO_EwrF2AC/view?usp=drivesdk

Thank you.

Regards

Coding boys.