IOWA STATE UNIVERSITY

Translational AI Center

Deep Dive into AI

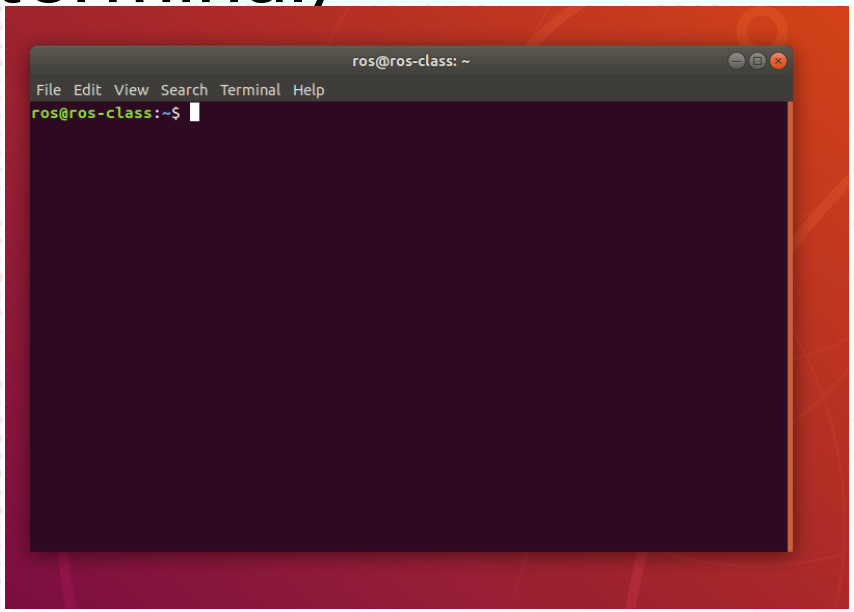
Python Programming

Objectives

- Introduction to Python Programming
  - Setting up Python environment
  - Python basics
  - Numpy, Scipy
  - Matplotlib

Terminal and terminal commands

- Terminal allows you to issue commands to execute (helps you perform a series of tasks programmatically)
- Opening a terminal (Right Click on the desktop and click on open terminal)



Few useful terminal commands

- Directory navigation etc.
  - ls (with arguments)
  - cd
  - pwd
  - Which/where
  - whoami
  - date
  - echo hello

- More resources can be found here:

  https://missing.csail.mit.edu/

Python

- Environment (which Python)
- Python 3 vs. Python 2 (now deprecated)
- Command line interface vs scripting
- Hello World! (in Python)

Python Environments - Hardwares

- Personal Machines
  - Windows/Mac/Linux
- Cloud
  - Google Colab
  - Github Codespaces
  - Code Studio
  - AWS/Azure/Google Cloud
  - Jetstream2
- HPC Clusters
  - Local clusters
  - National clusters

Python Environments - Software

- Native python
- Virtual environment
- Miniconda/Anaconda
- Singularity/Docker containers
- Virtual machines

Setting up environments

- Need Email ID for:
  - Vocareum
  - Jetstream2
  - Delta

Python

- Operators
  - Arithmetic Operators
  - Comparison Operators
  - Logical Operators
  - Assignment Operators
  - Bitwise Operators

Arithmetic Operators

Arithmetic operators are used to perform mathematical operations.

OPERATOR	DESCRIPTION	SYNTAX
+	Addition: adds two operands	x + y
-	Subtraction: subtracts two operands	x - y
*	Multiplication: multiplies two operands	x * y
/	Division (float): divides the first operand by the second	x / y
//	Division (floor): divides the first operand by the second	x // y
%	Modulus: returns the remainder when first operand is divided by the second	x % y
**	Exponentiation: Raise the first operand to the second	x ** y

Relational Operators

OPERATOR	MEANING	EXAMPLE
>	Greater than - True if left operand is greater than the right	x > y
<	Less than - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y

Logical Operators

OPERATOR	DESCRIPTION	SYNTAX
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if operand is false	not x

Bitwise Operators

In the table below: Let x = 10 (`0000 1010` in binary) and y = 4 (`0000 0100` in binary)

Operator	Meaning	Example
&	Bitwise AND	x& y = 0 (`0000 0000`)
|	Bitwise OR	x | y = 14 (`0000 1110`)
~	Bitwise NOT	~x = -11 (`1111 0101`)
^	Bitwise XOR	x ^ y = 14 (`0000 1110`)
>>	Bitwise right shift	x>> 2 = 2 (`0000 0010`)
<<	Bitwise left shift	x<< 2 = 40 (`0010 1000`)

Assignment Operators

OPERATOR	DESCRIPTION	SYNTAX
=	Assign value of right side of expression to left side operand	x = y + z
+=	Add AND: Add right side operand with left side operand and then assign to left operand	a+=b a=a+b
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand	a-=b a=a-b
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand	a*=b a=a*b
/=	Divide AND: Divide left operand with right operand and then assign to left operand	a/=b a=a/b
%=	Modulus AND: Takes modulus using left and right operands and assign result to left operand	a%=b a=a%b
//=	Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand	a//=b a=a//b
**=	Exponent AND: Calculate exponent (raise power) value using operands and assign value to left operand	a**=b a=a**b
&=	Performs Bitwise AND on operands and assign value to left operand	a&=b a=a&b
|=	Performs Bitwise OR on operands and assign value to left operand	a|=b a=a|b
^=	Performs Bitwise XOR on operands and assign value to left operand	a^=b a=a^b
>>=	Performs Bitwise right shift on operands and assign value to left operand	a>>=b a=a>>b
<<=	Performs Bitwise left shift on operands and assign value to left operand	a<<=b a=a<<b

Precedence of operations

The combination of values, variables, operators and function calls is termed as an expression. Python interpreter can evaluate a valid expression. To evaluate these types of expressions there is a rule of precedence in Python. It guides the order in which operation are carried out. The operator precedence in Python are listed in the following table. It is in descending order; upper group has higher precedence than the lower ones.

Operators	Meaning
()	Parentheses
**	Exponent
+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT
*, /, //, %	Multiplication, Division, Floor division, Modulus
+, -	Addition, Subtraction
<<, >>	Bitwise shift operators
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
==, !=, >, >=, <, <=, is, is not, in, not in	comparisons, Identity, Membership operators
not	Logical NOT
and	Logical AND
or	Logical OR

Python Data types

- Numbers
- Strings
- Lists
- Tuples
- Dictionaries

```
var = 382
```

```
var1 = 'Hello World!'
```

```
mylist = ['Rhino', 'Grasshopper', 'DoFlamingo', 'Bongo']
```

```
mytuple = ('Rhino', 'Grasshopper', 'DoFlamingo', 'Bongo')
```

```
room_num = {'john': 425, 'tom': 212}
```

```
message = "Good morning"
num = 85
pi = 3.14159
i = 0 + 1j
print(type(message))  # This will return string
print(type(n))  # This will return integer
print(type(pi))  # This will return float
print(type(i))  # This will return complex
```

TYPE	FORMAT	DESCRIPTION
Int	a = 10	SignedInteger
Long	a =345L	(L) Long integers, they can also be represented in octal and hexadecimal
Float	a = 45.67	(.) Floating point real values
Complex	a = 3.14j	(j) Contains integer in the range 0 to 255.

Python data types

- Numbers
- Strings
- Lists
- Tuples
- Dictionaries

```
var = 382
```

```
var1 = 'Hello World!'
```

```
mylist = ['Rhino', 'Grasshopper', 'DoFlamingo', 'Bongo']
```

```
mytuple = ('Rhino', 'Grasshopper', 'DoFlamingo', 'Bongo')
```

```
room_num = {'john': 425, 'tom': 212}
```

```
message = "Good morning"
num = 85
pi = 3.14159
i = 0 + 1j
print(type(message))  # This will return string
print(type(n))  # This will return integer
print(type(pi))  # This will return float
print(type(i))  # This will return complex
```

TYPE	FORMAT	DESCRIPTION
Int	a = 10	SignedInteger
Long	a =345L	(L) Long integers, they can also be represented in octal and hexadecimal
Float	a = 45.67	(.) Floating point real values
Complex	a = 3.14j	(j) Contains integer in the range 0 to 255.

Naming Restrictions

- Must begin with an alphabetic character (A -Z) or an underscore (_).
- Cannot contain a period(.), @, $, or %.
- Must be unique in the scope in which it is declared.
- Python is case sensitive. So “selection” and “Selection” are two different variables.

Scope, conditional and loops

```
food = 'spam'
if food == 'spam':
    print('Ummmm, my favorite!')
    print('I feel like saying it 100 times...')
    print(100 * (food + '! '))
```

```
>>> for i in range(5):
...     print('i is now:', i)
```

```
if True:                # This is always true
    pass                # so this is always executed, but it does nothing
else:
    pass
```

```
if choice == 'a':
    print("You chose 'a'.")
elif choice == 'b':
    print("You chose 'b'.")
elif choice == 'c':
    print("You chose 'c'.")
else:
    print("Invalid choice.")
```

```
while guess != name and pos < len(name):
    print("Nope, that's not it! Hint: letter ", end='')
    print(pos + 1, "is", name[pos] + ". ", end='')
    guess = input("Guess again: ")
    pos = pos + 1
    if pos == len(name) and name != guess:
        print("Too bad, you couldn't get it.  The name was", name + ".")
    else:
        print("\nGreat, you got it in", pos + 1,  "guesses!")
```

Python data types - summary

```
a = 4
b = 5
c = 6
d = 10


lists_ex1 = [
    [
        [0.2, 0.1, 0.2], 0.1, [0.4, 0.6, 0.9], ['me280', {'key1':'value1', 'key2':'value2'}],
        (0.4, 0.5, 0.6),
        set([0.2, 0.2, 0.2, 0.3, 0.5, 0.1, 0.6, 0.9]),
        ['another list 1', 'another list 2']
    ],
    [
        [a**i for i in range(6)],
        [b*i for i in range(10)],
        {a:b for a_i, b_i  in zip(range(b), range(a))},
        [(a_i,b_i) for a_i, b_i in zip(range(a), range(b))],
        [(idx, d_i) for idx, d_i in enumerate(reversed(range(d)))]
    ]
]
```

Functions

- Functions defined using def:

```
def func1(arg1, arg2, arg3, kwarg1='a', kwarg2='b', **kwargs):
    pass
```

```
def func2(*args, **kwargs):
    pass
```

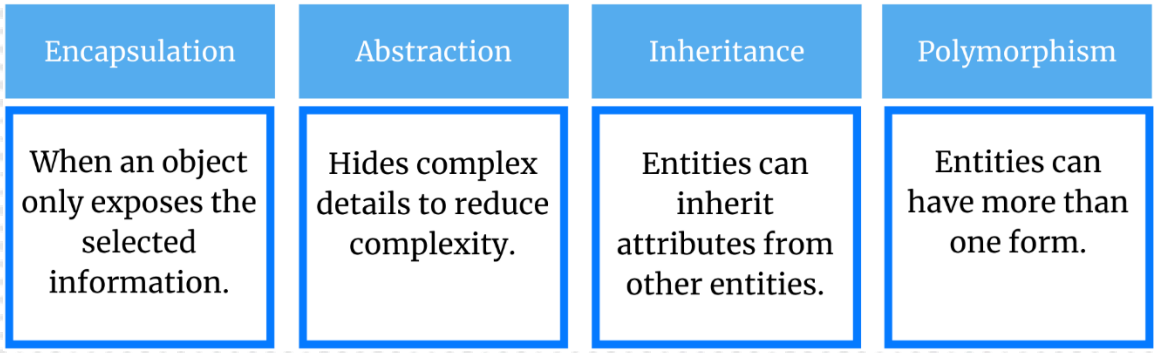
- Lambda functions:

```
func3 = lambda x: x**2

func4 = lambda x,y: (x+y)**2
```

Object Oriented Programming

- Key tenets of OOP:
  - Encapsulation
  - Inheritance
  - Abstraction
  - Polymorphism

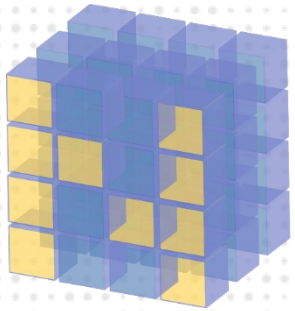


Defining the class (and dunder methods)

- __init__ method
- __repr__ method
- __len__ methods
- __get_item__ method
- __add__ etc.

Performing above operations on a list of values

Using Python Packages



NumPy

Pandas

SciPy

matplotlib

Numpy operations

- Comparison of manual implementation vs. numpy implementation
- Numpy is very optimized for performing python operations
- Vectors: $\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$
- Matrices: $\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$

Arrays, Basic Properties

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]],dtype=np.float32)
print a.ndim, a.shape, a.dtype
```

1. Arrays can have any number of dimensions, including zero (a scalar).
2. Arrays are typed: np.uint8, np.int64, np.float32, np.float64
3. Arrays are dense. Each element of the array exists and has the same type.

Arrays, creation

- np.ones, np.zeros
- np.arange
- np.concatenate
- np.astype
- np.zeros_like, np.ones_like
- np.random.random

Arrays, creation

- np.ones, np.zeros
- np.arange
- np.concatenate
- np.astype
- np.zeros_like, np.ones_like
- np.random.random

```
>>> np.ones((3,5),dtype=np.float32)
array([[ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.]], dtype=float32)
```

```
>>> np.zeros((6,2),dtype=np.int8)
array([[0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0]], dtype=int8)
```

Arrays, creation

- np.ones, np.zeros
- **np.arange**
- np.concatenate
- np.astype
- np.zeros_like, np.ones_like
- np.random.random

```
>>> np.arange(1334,1338)
array([1334, 1335, 1336, 1337])
```

Arrays, creation

- np.ones, np.zeros
- np.arange
- **np.concatenate**
- np.astype
- np.zeros_like, np.ones_like
- np.random.random

```
>>> A = np.ones((2,3))
>>> B = np.zeros((4,3))
>>> np.concatenate([A,B])
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
>>>
```

Arrays, creation

- np.ones, np.zeros
- np.arange
- **np.concatenate**
- np.astype
- np.zeros_like, np.ones_like
- np.random.random

```
>>> A = np.ones((4,1))
>>> B = np.zeros((4,2))
>>> np.concatenate([A,B], axis=1)
array([[ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.]])
```

Arrays, creation

- np.ones, np.zeros
- np.arange
- np.concatenate
- **np.astype**
- np.zeros_like, np.ones_like
- np.random.random

```
>>> A
array([[ 4670.5,  4670.5,  4670.5],
       [ 4670.5,  4670.5,  4670.5],
       [ 4670.5,  4670.5,  4670.5],
       [ 4670.5,  4670.5,  4670.5],
       [ 4670.5,  4670.5,  4670.5]], dtype=float32)
>>> print(A.astype(np.uint16))
[[4670 4670 4670]
 [4670 4670 4670]
 [4670 4670 4670]
 [4670 4670 4670]
 [4670 4670 4670]]
```

Arrays, creation

- np.ones, np.zeros
- np.arange
- np.concatenate
- np.astype
- **np.zeros_like, np.ones_like**
- np.random.random

```
>>> a = np.ones((2,2,3))
>>> b = np.zeros_like(a)
>>> print(b.shape)
```

Arrays, creation

- np.ones, np.zeros
- np.arange
- np.concatenate
- np.astype
- np.zeros_like, np.ones_like
- **np.random.random**

```
>>> np.random.random((10,3))
array([[ 0.61481644,  0.55453657,  0.04320502],
       [ 0.08973085,  0.25959573,  0.27566721],
       [ 0.84375899,  0.2949532 ,  0.29712833],
       [ 0.44564992,  0.37728361,  0.29471536],
       [ 0.71256698,  0.53193976,  0.63061914],
       [ 0.03738061,  0.96497761,  0.01481647],
       [ 0.09924332,  0.73128868,  0.22521644],
       [ 0.94249399,  0.72355378,  0.94034095],
       [ 0.35742243,  0.91085299,  0.15669063],
       [ 0.54259617,  0.85891392,  0.77224443]])
```

Arrays, danger zone

- Must be dense, no holes.
- Must be one type
- Cannot combine arrays of different shape

```
>>> np.ones([7,8]) + np.ones([9,3])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: operands could not be broadcast together
 with shapes (7,8) (9,3)
```

Shaping

```
a = np.array([1,2,3,4,5,6])
a = a.reshape(3,2)
a = a.reshape(2,-1)
a = a.ravel()
```

1. Total number of elements cannot change.
2. Use -1 to infer axis shape
3. Row-major by default (MATLAB is column-major)

Transposition

```
a = np.arange(10).reshape(5,2)
a = a.T
a = a.transpose((1,0))
```

np.transpose permutes axes.

a.T transposes the first two axes.

Saving and loading arrays

```
np.savez('data.npz', a=a)
data = np.load('data.npz')
a = data['a']
```

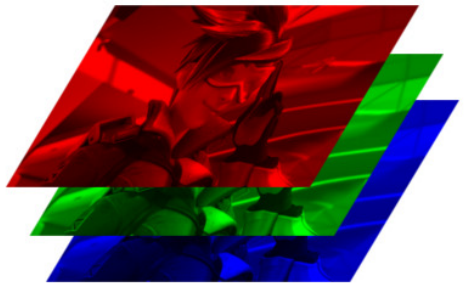
1. NPZ files can hold multiple arrays
2. np.savez_compressed similar.

Image arrays

Images are 3D arrays: width, height, and channels

Common image formats:

- height x width x RGB (band-interleaved)
- height x width (band-sequential)



Gotchas:

- Channels may also be BGR (OpenCV does this)
- May be [width x height], not [height x width]

Saving and Loading Images

SciPy: `skimage.io.imread,skimage.io.imsave`

height x width x RGB

PIL / Pillow: `PIL.Image.open, Image.save`

width x height x RGB

OpenCV: `cv2.imread, cv2.imwrite`

height x width x BGR

Mathematical operators

- Arithmetic operations are element-wise
- Logical operator return a bool array
- In place operations modify the array

```
>>> a
array([1, 2, 3])
>>> b
array([ 4,  4, 10])
>>> a * b
array([ 4,  8, 30])
```

Mathematical operators

- Arithmetic operations are element-wise
- **Logical operator return a bool array**
- In place operations modify the array

```
>>> a
array([[ 0.93445601,  0.42984044,  0.12228461],
       [ 0.06239738,  0.76019703,  0.11123116],
       [ 0.14617578,  0.90159137,  0.89746818]])
>>> a > 0.5
array([[ True, False, False],
       [False,  True, False],
       [False,  True,  True]], dtype=bool)
```

Mathematical operators

- Arithmetic operations are element-wise
- Logical operator return a bool array
- In place operations modify the arra

```
>>> a
array([[ 4, 15],
       [20, 75]])
>>> b
array([[ 2,  5],
       [ 5, 15]])
>>> a /= b
>>> a
array([[2, 3],
       [4, 5]])
```

Math, upcasting

Just as in Python and Java, the result of a math operator is cast to the more general or precise datatype.

uint64 + uint16 => uint64

float32 / int32 => float32

Warning: upcasting does not prevent overflow/underflow. You must manually cast first.

Use case: images often stored as uint8. You should convert to float32 or float64 before doing math.

Math, universal functions

Also called ufuncs

Element-wise

Examples:

- np.exp
- np.sqrt
- np.sin
- np.cos
- np.isnan

```
>>> a
array([[ 1,  4],
       [ 9, 16],
       [25, 36]])
>>> np.sqrt(a)
array([[ 1.,  2.],
       [ 3.,  4.],
       [ 5.,  6.]])
```

Indexing

```
x[0,0]    # top-left element
x[0,-1]   # first row, last column
x[0,:]    # first row (many entries)
x[:,0]    # first column (many entries)
```

Notes:

- Zero-indexing
- Multi-dimensional indices are comma-separated (i.e., a tuple)

Indexing, slices and arrays

```
I[1:-1,1:-1]       # select all but one-pixel
border
I = I[:,:,::-1]    # swap channel order
I[I<10] = 0        # set dark pixels to black
I[[1,3], :]        # select 2nd and 4th row
```

1. Slices are **views**. Writing to a slice overwrites the original array.
2. Can also index by a list or boolean array.

Python Slicing

Syntax: start:stop:step

```
a = list(range(10))
a[:3] # indices 0, 1, 2
a[-3:] # indices 7, 8, 9
a[3:8:2] # indices 3, 5, 7
a[4:1:-1] # indices 4, 3, 2 (this one is tricky)
```

Axes

```
a.sum() # sum all entries
a.sum(axis=0) # sum over rows
a.sum(axis=1) # sum over columns
a.sum(axis=1, keepdims=True)
```

1. Use the axis parameter to control which axis NumPy operates on
2. Typically, the axis specified will disappear, keepdims keeps all dimensions

Broadcasting

```
a = a + 1 # add one to every element
```

When operating on multiple arrays, broadcasting rules are used.

Each dimension must match, from right-to-left

1. Dimensions of size 1 will broadcast (as if the value was repeated).
2. Otherwise, the dimension must have the same shape.
3. Extra dimensions of size 1 are added to the left as needed.

Broadcasting example

Suppose we want to add a color value to an image

a.shape is 100, 200, 3

b.shape is 3

a + b will pad b with two extra dimensions so it has an effective shape of 1 x 1 x 3.

So, the addition will broadcast over the first and second dimensions.

Broadcasting failures

If a.shape is 100, 200, 3 but b.shape is 4 then a + b will fail. The trailing dimensions must have the same shape (or be 1)

Matplotlib

```
import matplotlib.pyplot as plt

xs = [1,2,3,4,5]
ys = [x**2 for x in xs]

plt.plot(xs, ys)
```

```
import matplotlib.pyplot as plt

xs = range(-100,100,10)
x2 = [x**2 for x in xs]
negx2 = [-x**2 for x in xs]

plt.plot(xs, x2)
plt.plot(xs, negx2)
plt.xlabel("x")
plt.ylabel("y")
plt.ylim(-2000, 2000)
plt.axhline(0) # horiz line
plt.axvline(0) # vert line
plt.savefig("quad.png")
plt.show()
```

Incrementally modify the figure.

Save your figure to a file

Show it on the screen

```
def myplot(xs, ys, description):
  plt.plot(xs, ys, linewidth=2, color='green', linestyle='-', marker='s', label=description)

def setup_plot():
  plt.xlabel("x")
  plt.ylabel("y")
  plt.axhline(0,linestyle=':',color='red')
  plt.axvline(0,linestyle=':',color='red')

def finish_plot():
  plt.legend()
  plt.show()

setup_plot()
myplot(xs,x2,"x**2")
finish_plot()

setup_plot()
myplot(xs,negx2,"-x**2")
finish_plot()
```

We can group these options into functions as usual, but remember that they are operating on a global, hidden variable