

EGM722 – Programming for GIS and Remote Sensing

Week 1, Part 1: Why programming?

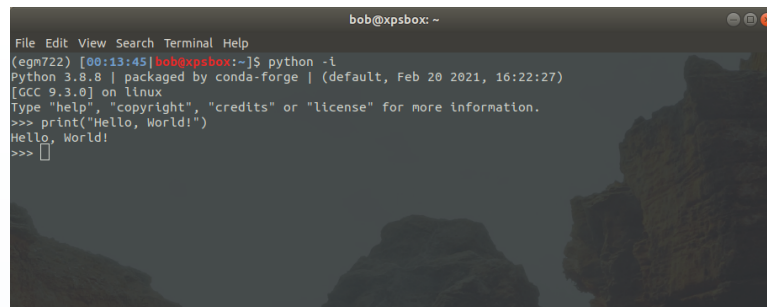
1. A brief introduction to programming using python
2. Built-in types
3. Controlling Flow
4. Functions
5. A brief introduction to git

Why programming?

- In GIS and Remote Sensing, we often repeat the same tasks over and over
 - e.g., processing new/different data for different areas
- These tasks often form a workflow
 - What happens when you need to re-do a single step in the middle or beginning of the workflow?
 - What happens when someone else needs to do the same steps that you did?
 - What happens if we're lazy and don't want to do the same tasks over and over?
- Computers: really good at repeating the same tasks over and over and over and over and over and over and over and over and...
 - But (for now), we have to give them instructions: [programming](#)

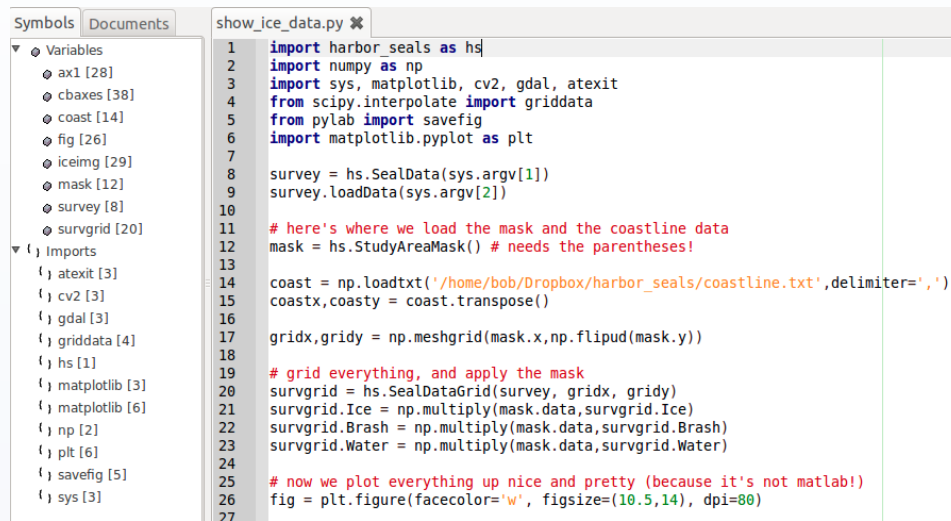
What is python?

- An **interpreted, high-level** language
- Python interpreter:
 - Reads code
 - Translates it
 - Executes it
- Run in two ways:
 - **Interactive** mode
 - **Script** mode



```

bob@xpsbox: ~
File Edit View Search Terminal Help
(egm722) [00:13:45] bob@xpsbox:~$ python -i
Python 3.8.8 | packaged by conda-forge | (default, Feb 20 2021, 16:22:27)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hello, World!")
Hello, World!
>>>
  
```



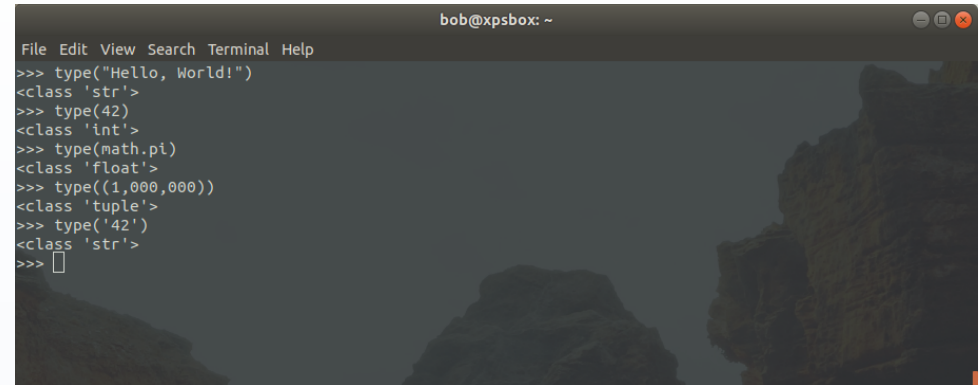
```

Symbols Documents show_ice_data.py
Variables
  ax1 [28]
  cbaxes [38]
  coast [14]
  fig [26]
  iceimg [29]
  mask [12]
  survey [8]
  survgrid [20]
Imports
  atexit [3]
  cv2 [3]
  gdal [3]
  griddata [4]
  hs [1]
  matplotlib [3]
  matplotlib [6]
  np [2]
  plt [6]
  savefig [5]
  sys [3]

1 import harbor_seals as hs
2 import numpy as np
3 import sys, matplotlib, cv2, gdal, atexit
4 from scipy.interpolate import griddata
5 from pylab import savefig
6 import matplotlib.pyplot as plt
7
8 survey = hs.SealData(sys.argv[1])
9 survey.loadData(sys.argv[2])
10
11 # here's where we load the mask and the coastline data
12 mask = hs.StudyAreaMask() # needs the parentheses!
13
14 coast = np.loadtxt('/home/bob/Dropbox/harbor_seals/coastline.txt', delimiter=',')
15 coastx, coasty = coast.transpose()
16
17 gridx, gridy = np.meshgrid(mask.x, np.flipud(mask.y))
18
19 # grid everything, and apply the mask
20 survgrid = hs.SealDataGrid(survey, gridx, gridy)
21 survgrid.Ice = np.multiply(mask.data, survgrid.Ice)
22 survgrid.Brash = np.multiply(mask.data, survgrid.Brash)
23 survgrid.Water = np.multiply(mask.data, survgrid.Water)
24
25 # now we plot everything up nice and pretty (because it's not matlab!)
26 fig = plt.figure(facecolor='w', figsize=(10.5, 14), dpi=80)
27
  
```

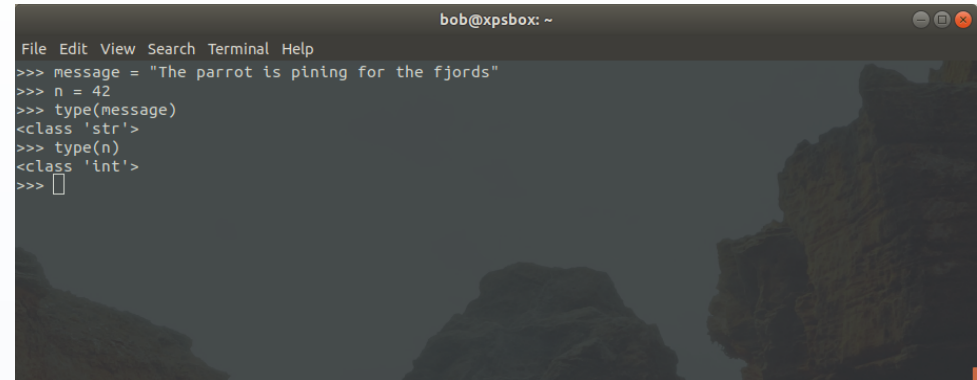
The world is filled with objects

- Python is an **object-oriented** programming language
- **Object**: the basic “thing” that python works with
- Objects have:
 - **type**
 - properties
 - **methods**



```
File Edit View Search Terminal Help
>>> type("Hello, World!")
<class 'str'>
>>> type(42)
<class 'int'>
>>> type(math.pi)
<class 'float'>
>>> type((1,000,000))
<class 'tuple'>
>>> type('42')
<class 'str'>
>>> 
```

- A **variable** is a name that refers to an object
- “Like a box in the computer’s memory where you can store a single value” (Swiegart, 2020)
- If we want to save values to use later, have to store them in a variable
- To create variables, use **assignment statements**
 - NB: ‘=’ does not mean ‘equal to’!



```

bob@xpsbox: ~
File Edit View Search Terminal Help
>>> message = "The parrot is pining for the fjords"
>>> n = 42
>>> type(message)
<class 'str'>
>>> type(n)
<class 'int'>
>>> 
  
```

Naming variables

- Good practice:
 - Choose meaningful names
 - Names **must** begin with a letter
 - Can contain underscores
- Bad practice:
 - Names cannot contain illegal characters (@, !, etc.)
 - Cannot be a protected keyword (**and**, **for**, **if**, etc.)
 - Try not to overwrite built-in types/classes (**list**, **int**, etc.)

- We use **operators** to perform some kind of computation
- Examples:
 - `+`: addition (**concatenation** for strings and lists)
 - `-`: subtraction
 - `*`: multiplication (also works for strings, lists)
 - `/`: division
 - In python 3, normal division (e.g., $4/5 = 0.8$)
 - In python 2, **floor** division (e.g., $4/5 = 0$)
 - python 2 is (mostly) gone now, unless you use ArcMap
 - `**`: exponentiation (NB: `^` is a separate operator)
 - `%`: modular arithmetic

Order of operations

- Python follows PEMDAS:
 - Parentheses
 - Exponentiation
 - Multiplication/Division
 - Addition/Subtraction
 - Operators with same precedence are evaluated left to right
- $2*(3 - 1)$
 - $(1+1)**(5-2)$
 - $2**1+1$
 - $3*1**3$
 - $2*3-1$
 - $6+4/2$

- An expression is a combination of objects, variables, and operators:
 - 42
 - x
 - $x+42$
- A statement is a unit of code the interpreter can execute
 - Assignment statements (e.g., $x = 42$)
 - return, pass statements

- Programming is giving a computer instructions to execute tasks
- Python: one language we can use to give a computer instructions
- Python uses objects to carry out computations and other tasks

- Automate the Boring Stuff with Python (2nd ed.)
- Automate the Boring Stuff with Python [[youtube](#)]
- [Learnpython.org](#)
- Beginner's Guide to Python [[python.org](#)]
- Python documentation (3.8.8) [[python.org](#)]