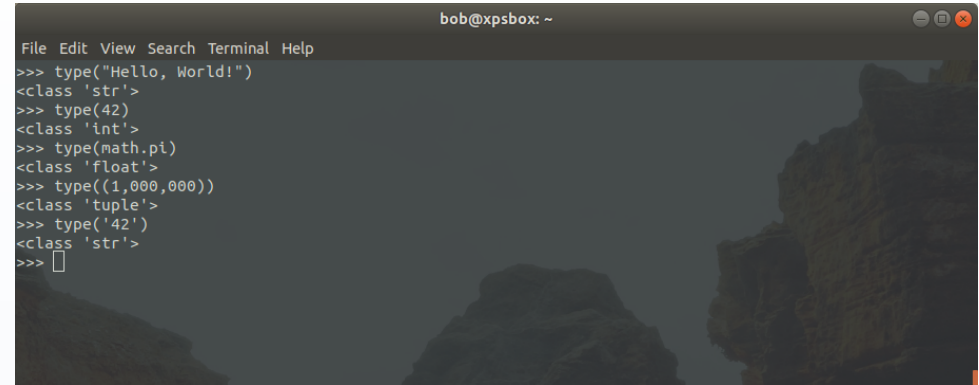


# EGM722 – Programming for GIS and Remote Sensing

Week 1, Part 2: Built-in types

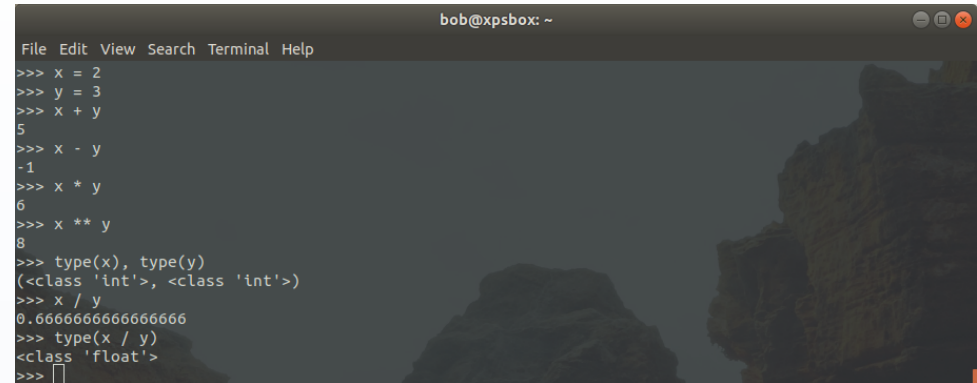
# Recall: The world is filled with objects

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

A screenshot of a terminal window titled 'bob@xpsbox: ~'. The terminal shows a series of Python commands and their outputs, demonstrating the 'type' function. The background of the terminal window features a dark, rocky landscape.

```
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'>
>>> 
```

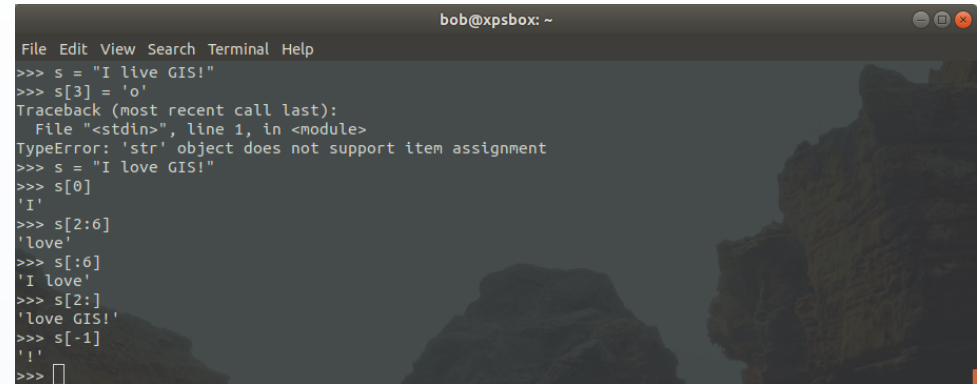
- Python has three main numeric types:
  - integers
  - floating-point (decimal) numbers
  - complex numbers
- When combining types, result uses the 'wider' definition
  - e.g., `int + float => float`
  - `float + complex => complex`



```

bob@xpsbox: ~
File Edit View Search Terminal Help
>>> x = 2
>>> y = 3
>>> x + y
5
>>> x - y
-1
>>> x * y
6
>>> x ** y
8
>>> type(x), type(y)
(<class 'int'>, <class 'int'>)
>>> x / y
0.6666666666666666
>>> type(x / y)
<class 'float'>
>>>
  
```

- Python handles text as **strings**
- Strings are **immutable** arrays of text characters
  - Cannot modify/change elements
- Individual characters are located at an **index**
  - NB: in python, arrays start from 0
- To access multiple indices, can **slice**: `s[n:m]`
  - If we omit `n`, starts from beginning
  - If we omit `m`, goes to end
  - Can also use negative numbers



```

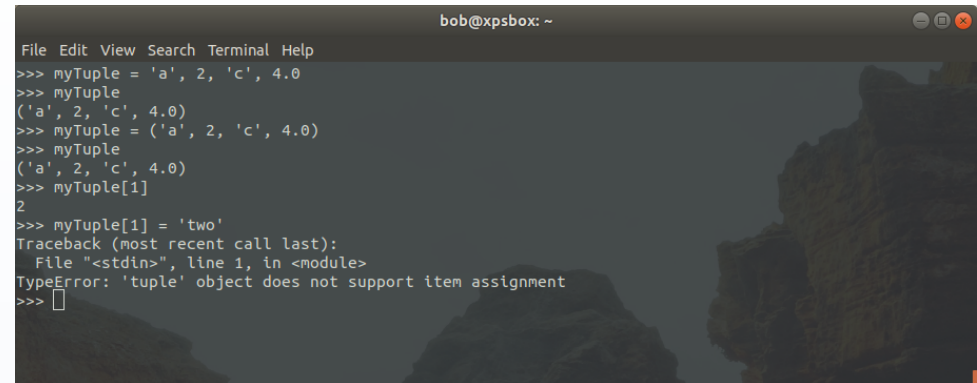
bob@xpsbox: ~
File Edit View Search Terminal Help
>>> s = "I love GIS!"
>>> s[3] = 'o'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> s = "I love GIS!"
>>> s[0]
'I'
>>> s[2:6]
'love'
>>> s[:6]
'I love'
>>> s[2:]
'love GIS!'
>>> s[-1]
'!'
>>> 
  
```

- A list is a sequence of values called elements or items
- Can create with `[ ]`, `list()`
- Elements can be any type
  - Can also be mixed types
  - Can be **nested** (a list of lists)
- lists are **mutable**

```

bob@xpsbox: ~
File Edit View Search Terminal Help
>>> myList = [10, '42', 13.0, [10, 20]]
>>> myList
[10, '42', 13.0, [10, 20]]
>>> type(myList[2])
<class 'float'>
>>> type(myList[1])
<class 'str'>
>>> type(myList[3])
<class 'list'>
>>> myList[3][0]
10
>>> myList[1] = 'forty-two'
>>> myList
[10, 'forty-two', 13.0, [10, 20]]
>>> 
  
```

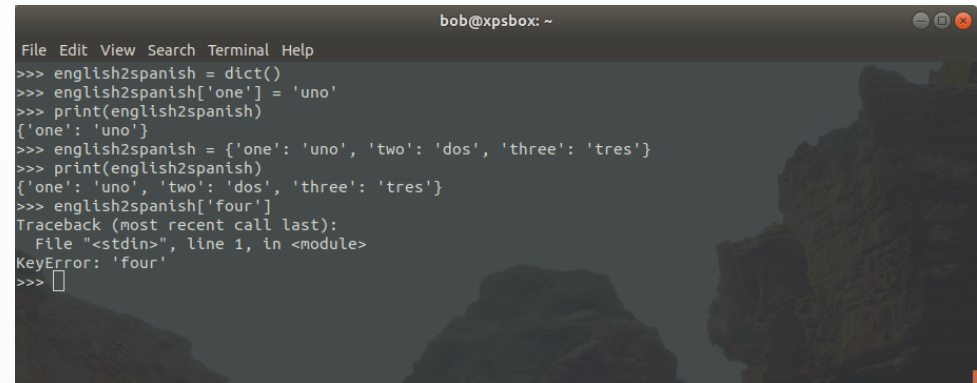
- **tuples** work like lists:
  - Can contain multiple types
  - Index using [ ]
  - Can be nested
- Create with commas, ( )
- Tuples are **immutable**



```

bob@xpsbox: ~
File Edit View Search Terminal Help
>>> myTuple = 'a', 2, 'c', 4.0
>>> myTuple
('a', 2, 'c', 4.0)
>>> myTuple = ('a', 2, 'c', 4.0)
>>> myTuple
('a', 2, 'c', 4.0)
>>> myTuple[1]
2
>>> myTuple[1] = 'two'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> 
  
```

- A **dictionary** is like a list, but more general
  - List indices: **integers**
  - Dictionary indices: almost anything
- Create using `dict()`, `{ }`
- Dictionaries **map** indices (**keys**) to **values**: key-value pair



```

bob@xpsbox: ~
File Edit View Search Terminal Help
>>> english2spanish = dict()
>>> english2spanish['one'] = 'uno'
>>> print(english2spanish)
{'one': 'uno'}
>>> english2spanish = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
>>> print(english2spanish)
{'one': 'uno', 'two': 'dos', 'three': 'tres'}
>>> english2spanish['four']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'four'
>>> 
  
```

- Python has a number of built-in types that we can use
- Most common (for now) are numbers and sequences
- Sequences can be indexed, sliced to access smaller parts of the whole
- Lists can be changed; strings and tuples cannot