Python

Aliasing in Python

Mutable objects and aliasing

Mutable objects in Python can be subject to *aliasing.*

Mutable objects and aliasing

Mutable objects in Python can be subject to *aliasing.*

```
>>> a = [[0, 1, 2], [0, 1, 2]]
>>> print(a)
[[0, 1, 2], [0, 1, 2]]
```

Mutable objects and aliasing

Mutable objects in Python can be subject to *aliasing.*

```
>>> a = [[0, 1, 2], [0, 1, 2]]
>>> print(a)
[[0, 1, 2], [0, 1, 2]]
>>> b = a[1]
>>> print(b)
[0, 1, 2]
```

Mutable objects and aliasing

Mutable objects in Python can be subject to *aliasing.*

```
>>> a = [[0, 1, 2], [0, 1, 2]]
>>> print(a)
[[0, 1, 2], [0, 1, 2]]
>>> b = a[1]
>>> print(b)
[0, 1, 2]
>>> b[1] = 'hello'
```

Mutable objects and aliasing

Mutable objects in Python can be subject to *aliasing.*

```
>>> a = [[0, 1, 2], [0, 1, 2]]
>>> print(a)
[[0, 1, 2], [0, 1, 2]]
>>> b = a[1]
>>> print(b)
[0, 1, 2]
>>> b[1] = 'hello'
>>> print(b)
[0, 'hello', 2]
```

Mutable objects and aliasing

Mutable objects in Python can be subject to *aliasing.*

```
>>> a = [[0, 1, 2], [0, 1, 2]]
>>> print(a)
[[0, 1, 2], [0, 1, 2]]
>>> b = a[1]
>>> print(b)
[0, 1, 2]
>>> b[1] = 'hello'
>>> print(b)
[0, 'hello', 2]
>>> print(a)
```

Mutable objects and aliasing

Mutable objects in Python can be subject to *aliasing.*

```
>>> a = [[0, 1, 2], [0, 1, 2]]
>>> print(a)
[[0, 1, 2], [0, 1, 2]]
>>> b = a[1]
>>> print(b)
[0, 1, 2]
>>> b[1] = 'hello'
>>> print(b)
[0, 'hello', 2]
>>> print(a)
[[0, 1, 2], [0, 'hello', 2]]
```

Aliasing - why?

Why would aliasing be useful in a programming language?

- Efficiency - especially with "big" arrays
- Sometimes you want to be able to assign a variable to a sub-component of another variable (such as a list, array, dictionary or more complex object) - and to *change* it.

An example? (1)

Imagine you have a massive data array of temperatures:

Latitude \ Longitude	10°E	20°E
60°N	14	13
50°N	16	15
40°N	34	21

```
>>> temps = [[14, 16, 34], [13, 15, 21]]
```

Each sub-list contains temperatures for a given longitude.

Let's assign a variable to the first sub-list because we want to process/modify it:

```
>>> temp_lon_1 = temps[0]
```

An example? (2)

```
>>> print(temp_lon_1)
[14, 16, 34]
```

Let's change some values and see the effect on the overall variable *temps*.

```
>>> temp_lon_1[:2] = [15, 17]
>>> print(temp_lon_1)
[15, 17, 34]

>>> print(temps)
[[15, 17, 34], [13, 15, 21]]
```

Avoiding aliasing

If I know I don't want to create an alias what can I do?

Avoiding aliasing

If I know I don't want to create an alias what can I do?

Python's **`copy.deepcopy`** function will make a full copy of an object to want to replicate.

```
>>> import copy
>>> new_obj = copy.deepcopy(my_obj)
```