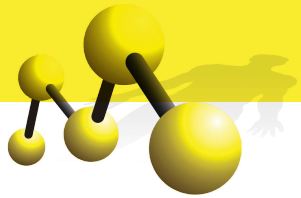
Python's Odds and Ends

Luis Pedro Coelho

Programming for Scientists

September 16, 2012

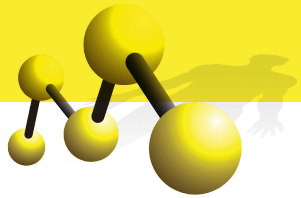
Python So Far



Python

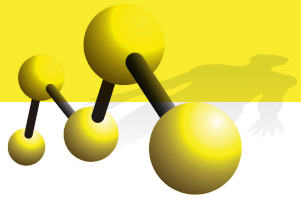
1. Basic types: int, float, list
2. Control flow: for, while, if, else, elif

List Indexing



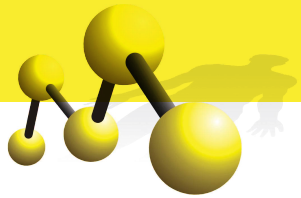
```
students = ['Luis','Rita','Sabah','Grace']
print students[0]
print students[1:2]
print students[1:]
print students[-1]
print students[-2]
```

Tuples (I)



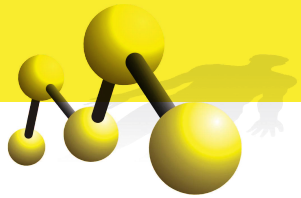
```
A = (0,1,2)
B = (1,)

print A[0]
print len(B)
```

Tuples (II)



Tuples are like immutable lists.

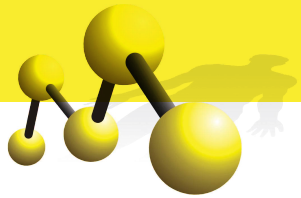
Dictionaries



- Dictionaries are associative arrays.

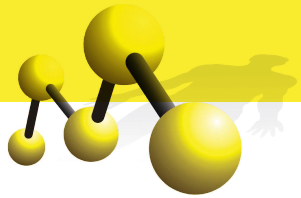
```
gene2ensembl = {}
gene2ensembl['SMAD9'] = 'ENSG00000120693'
gene2ensembl['ZNF670'] = 'ENSG00000135747'

print gene2ensembl['SMAD9']
```

Dictionary Methods



```
gene2expression = {
    'SMAD9' : 12.3,
    'ZNF670' : 4.3,
}

print len(gene2ensembl)
print gene2ensembl.keys()
```

Set Type

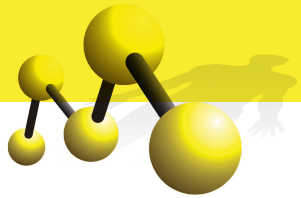


```
numbers = set([1,2,5])
print 3 in numbers
numbers.add(4)
print numbers
numbers.add(1)
print numbers
print numbers | set(['Rita'])
print numbers - set([2,3])
```

Output:

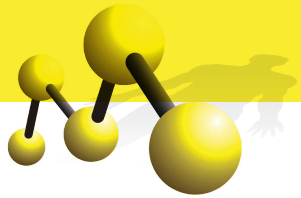
```
False
set([1, 2, 4, 5])
set([1, 2, 4, 5])
set([1, 2, 4, 5, 'Rita'])
set([1, 4, 5])
```

Frozenset Type

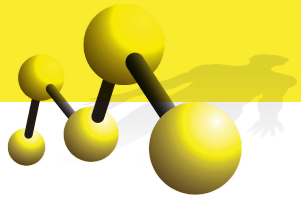


```
numbers = frozenset([1,2,5])

print 3 in 5 # False
print 2 in 5 # True

numbers.add(1) # ERROR!!
```

What's up With Immutability?

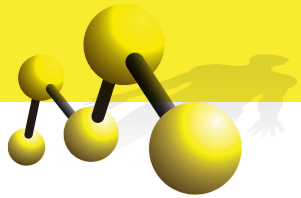


What's up With Immutability?



You can only use immutable objects as dictionary keys!

Complex Numbers

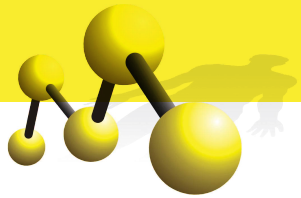


```
A = 1+1j
print A**2
print A**4
```

prints

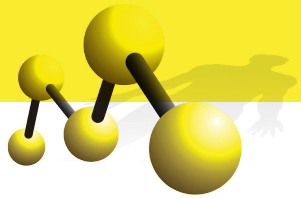
```
2j
(-4+0j)
```

None object



None

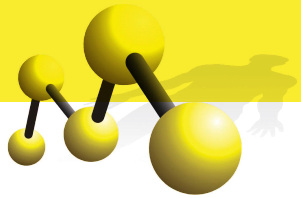
Object Identity



Object Identity

- A is B
- A is not B
- id(obj)

List Comprehensions

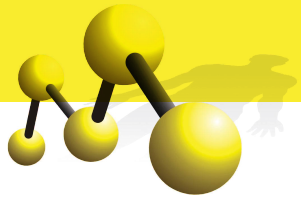


```
name = [ <expr> for <name> in <sequence> if <condition> ]
```

maps to

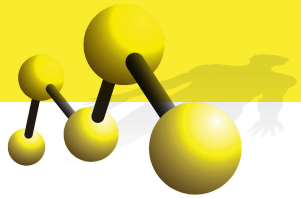
```
name = []
for <name> in <sequence>:
    if <condition>:
        name.append(<expr>)
```

List Comprehensions Example



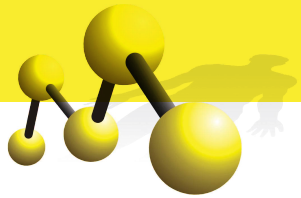
```
squares = [x*x for x in xrange(1,20)]
evensquares = [x*x for x in xrange(1,20) if (x%2) == 0]

squares = []
for x in xrange(1,20):
    squares.append(x*x)

evensquares = []
for x in xrange(1,20):
    if (x%2) == 0:
        evensquares.append(x*x)
```

Functions



```
def max(arg0,*args):
    '''
    M = max(arg0,arg1,...)

    Returns the maximum of its arguments
    '''
    M = arg0
    for val in args:
        if val > M:
            M = val
    return M
```

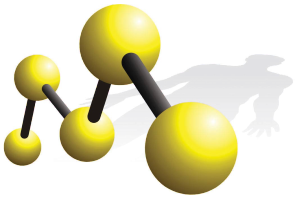
Multiple Assignment



```
A, B = 1, 2
```

Assign multiple elements at once.

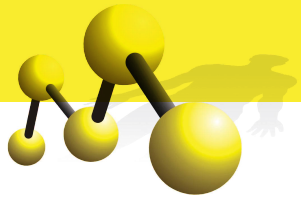
Multiple Assignment to Return Multiple Arguments



```
def stats(values):
    '''...'''
    return mean(values), std(values)

...
values = ...
props = stats(values)
mu, std = stats(values)
```

```
def greet(name, greeting='Hello'):
    '''
    greet(name, greeting='Hello')

    Greets person by name

    Parameters
    ----------
    name: str
        Name
    greeting: str, optional
        Greeting to use
    '''
    print greeting, name

ret = greet('World')
```

Sequences



```
for value in sequence:
    ...
```

Sequences

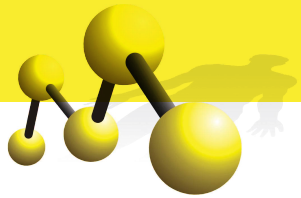
- Lists
- Tuples
- Sets & Frozensets
- Dictionaries
- ...

Generators

Generator: "Function"-like Sequence

```
def xrange(start,stop=None,step=None):
    '''
    xrange([start,]stop[,step]) -> xrange object

    Like range, but instead of a list, returns...
    '''
    if stop is None and step is None:
        stop = start
        start = 0
        step = 1
    elif step is None:
        step = 1

    while start < stop:
        yield start
        start += step
```

Generators



- Generators are similar to functions, but generate a sequence.
- Functions use return, generators use yield.