Excercise

- Write down the main topic/title of the last five classes on Python
- For each class, what were the main ideas
  - Outline each class.
  - Put in details
  - What examples were done?
- Do this alone, then with your neighbor.
- Create your own review notes (like these slides).
  - First “recall” then look up to fill in.

Class 14: Variables, scope, tuples, dictionaries

- Variables *tag* objects
  - id(x) → memory location of x
  - type(x) → object type of x
- Immutable objects: numbers, strings, tuples
- Mutable objects: lists, dictionaries, np arrays
- Scope, namespace

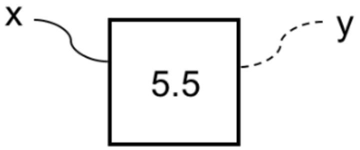
Variables, scope, tuples, dictionaries

Ploting, numpy file i/o

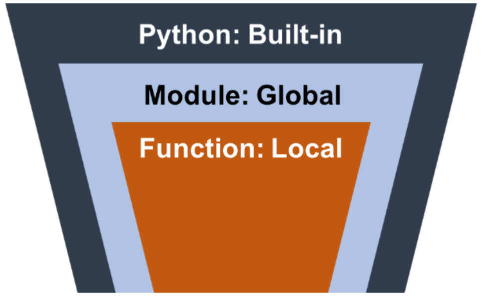
Modules and classes

Integration

x = 5.5 → y = 5.5



5.5 is its own object;
x, and y are tags
on that object




```python
g = 9.81
def f(m):
    return m*g     # use global g
f(10)
```

```
98.10000000000001
```

```python
g = 9.81
def f(m):
    g = 1            # new local g
    return g*m
print(g, f(20))    # same global g
```

```
9.81 20
```

```python
def f(x):
    x[2] = 55555

x = [0,1,2]
f(x)
print(x)     # x changed through func arg
```

```
[0, 1, 55555]
```

Class 14: Variables, scope, tuples, dictionaries

- Variables, scope, tuples, dictionaries
- Ploting, numpy file i/o
- Modules and classes
- Integration

```python
ages = {'alice': 15, 'john': 88, 'bill': 22}

for name in ages:
    print(ages[name])

print( 'dan' in ages )
```

```
15
88
22
False
```

- Tuples
  - Like lists, but immutable.
  - Define as: t = (3.14, 8314, 101325, 88)
  - Use comma for single element: t = (3.14,)
  - Unpacking: x,y = (5, 6)
  - Use to swap variables: x = 7, y = 8; x,y = (y,x)
  - Access elements as t[0], or t[i], or t[i+5], t[-1], etc.
- Dictionaries
  - Like lists, but indexed by a key that we set, instead of indexed by integer
  - Define as t = {key:value, key:value, etc.}
  - ages = {‘alice’: 15, ‘john’: 88, ‘bill’: 22, ‘sue’: 19}
  - Access as ages[‘sue’]
  - Functions: ages.keys(), ages.values(), ages.items()
  - Functions: del ages[‘john’], ages.pop[‘john’], ages.popitem()
  - Initialize: ages = {}
  - Add data ages[some_key] = some_value

Class 15:Plotting, file i/o


```python
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```python
plt.plot(x,y)
```

```python
plt.plot(x,y, 'ko-')
plt.xlabel('The x-axis label')
plt.ylabel('The y-axis label')
plt.title('Some title')
plt.legend(['my curve']);
```

```python
plt.subplots(2,1,1)
plt.plot(x,y)
plt.xlabel('x')
plt.xlabel('y')

plt.subplots(2,1,2)
plt.plot(x,y)
plt.xlabel('x')
plt.xlabel('y')

plt.tight_layout()
```

```python
plt.plot(x,     y+0.5,          '--', color='black', linewidth=2)
plt.plot(x[1:], 1/x[1:]**0.2, 'x-', color='blue',  markersize=6)

plt.yscale('log')
plt.xlim([0,6])
plt.ylim([0.1,10])
plt.xlabel(r'With Symbols: $\alpha_{s}\beta^{s}$', fontsize=20)
plt.ylabel('some label', fontsize=20)
plt.legend(['curve 1', 'curve2'], fontsize=16, frameon=False, loc='upper left');
```

Class 15:Plotting, file i/o

Variables, scope, tuples, dictionaries

Ploting, numpy file i/o

Modules and classes

Integration

Numpy

- `np.savetxt`
- `np.loadtxt`
- `np.column_stack`

Create some data

```
x     = np.linspace(0,10,25)
y_exp = x**0.6 + (np.random.rand(25) -0.5)
y_1   = x**0.6
y_2   = np.exp(-0.2*x)*np.cos(2*x)*2
```

Create a single 2-D array (matrix) for saving to a file

```
data = np.column_stack([x,y_exp, y_1, y_3])
print(data)
```

Save to file

```
np.savetxt("data.dat", data, fmt='%10.5f', header="x, y_exp, y_1, y_2")
```

Load from file

```
loaded_data = np.loadtxt("data.dat")
print(loaded_data)
```

Class 16: Modules, Classes

Variables, scope, tuples, dictionaries
Ploting, numpy file i/o
Modules and classes
Integration

`import` statement

- Import options for numpy:

```
import numpy as np
import numpy
import numpy as numnum
from   numpy import sin, cos
from   numpy import *
```

- Modules
  - Save python code in a textfile as code.py
    - Put in the same folder as you are using it from.
    - Otherwise do this:
    - Import sys
    - Sys.path.append(‘path/to/folder/containing/the/module
  - Import the modue: import code as c
  - Use variables and functions:
    - x = c.some_variable
    - y = c.some_function()

Class 16: Modules, Classes

Variables, scope, tuples, dictionaries
Ploting, numpy file i/o
Modules and classes
Integration

- Classes

```python
class cube():

    def set_side(self, side):
        self.L = side

    def volume(self):
        return self.L**3

    def area(self):
        return 6 * self.L**2
```

```python
c1 = cube()             # create a cube object
c2 = cube()             # create another cube object

c1.set_side(2)             # set side length
c2.set_side(3)             # set side length

print('Sides   =', c1.L,          c2.L)
print('Areas   =', c1.area(),     c2.area())
print('Volumes =', c1.volume(),   c2.volume())
```

```
Sides   = 2 3
Areas   = 24 54
Volumes = 8 27
```

Class 17: Integration

Variables, scope, tuples, dictionaries
Ploting, numpy file i/o
Modules and classes
Integration

- Examples
  - Simple numerical integration (rectangle)
    - Sum of rectangle areas
  - Trapazoid integration (N trapazoids)

$$I = -\frac{\Delta x}{2}(f_0 + f_N) + \Delta x \sum_{i=0}^{N} f_i.$$

  - Adaptive, recursive trapazoid integration
- Python quad function

```python
from scipy.integrate import quad

def f(x, param):
    return x**2 + param

xlo   = 1
xhi   = 2
param = 3
I = quad(f, xlo, xhi, args=(param,))[0]
```

Class 18: Solving Nonlinear Equations

Solving nonlinear equations
Args/tuple expansion, nonlinear HW, examples
Interpolation
Curve fitting

- One equation and one unknown
  - Put in f(x)=0 form.

```python
import numpy as np
from scipy.optimize import fsolve

def f(x):
    return x**2 - 5

xg = 1
x  = fsolve(f, xg)
```

- Add args if needed (as in quad)

Class 18: Solving Nonlinear Equations

Solving nonlinear equations
Args/tuple expansion, nonlinear HW, examples
Interpolation
Curve fitting

- Multiple equations in multiple unknowns
  - Pass in an array of unknowns
  - Return an array of equation values

```
import numpy as np
from scipy.optimize import fsolve

def hg(yz):

    y = yz[0]                 # recover the vars for convenience
    z = yz[1]

    h = y + 2*z               # compute the function values
    g = np.sin(y)/z

    return np.array([h, g])   # return array of function values

#----------------------------------

yz_g = np.array([1,2])
yz = fsolve(hg, yz_g)
y = yz[0]
z = yz[1]
```

  - Note, use obvious names.
  - Note the structure: recover, find equations, return array