# Study Exercise

- Make a list of all python functions/variables/code we have seen.
- Consider the items in the list:
  - What is the basic idea?
  - What further details?
  - What are some “gotchas” or caveats?

# Study Excercise

- Write down the main topic/title of each class
  - Try without the schedule, then with it
- For each class, what were the main ideas
  - Outline each class.
  - Put in details
  - What examples were done?
- Do this alone, then with others.
- Create your own review notes (like these slides).
  - First “recall” then look up to fill in.

# Study Exercise

- Go back through the homework.
- Read the problem statements and think through the problems
  - What functions are needed?
  - What is the approach used?
  - What will the code look like
- Outline the the solution
- Don’t spend too much time trying to rework problems, but get the essense of the problem and try to do it from memory.
- Afterward, look at the homework solution provided and compare your approach.

# Study Exercise

- Write out simple examples of code features and how/why we used them.
  - Loops
  - Functions
  - Conditionals
  - Variables
  - Etc.
- Do the same for key numerical tools considered.
- “Play professor,” what would you put on the exam? Try inventing your own problems.

# Class 24: Rate Equations

- Rate equations
- Symbolic math
- Widgets
- Python-Excel interface

- from scipy.integrate import odeint

$$\frac{dy}{dt} = f(y,t) \qquad y(0) = y_0$$

- f(y,t) is the “right hand side function” or the rate function.
- f(y,t) depends on y and t in general, but the actual expression often doesn’t include t.
- For multiple equations y is a vector of “variables”, f is a vector of functions
- Solve for y(t). Solution will be an array of t and an array of y

$$\frac{dy}{dt} = -2y + 3$$

$$y(0) = 1$$

```
def f(y,t):
    return -2*y + 3

y0 = 1
t  = np.linspace(0,5,100)
y  = odeint(f, y0, t)
```

$$\frac{dv}{dt} = g$$

$$\frac{dx}{dt} = v$$

$$y(0) = x(0) = 0$$

```
def f(vx,t):
    v = vx[0]
    x = vx[1]
    dvdt = 9.81
    dxdt = v
    return np.array([dvdt, dxdt])

xy0 = np.array([0, 0])
t  = np.linspace(0,5,100)
y  = odeint(f, xy0, t)
```

# Class 25: Symbolic Math

Rate equations

Symbolic math

Widgets

Python-Excel interface

```
import sympy as sp
sp.init_printing()
from IPython.display import display
```

```
x,y,z = sp.symbols('x, y, z')                  # set symbols

ex = x**2 + y**2 + z                           # create an expression

display(ex)                                    # display expression (better than print)

ex2 = ex.subs(y+1,x)                           # substitute y+1 for x
ex3 = ex.subs( [(y+1,x), (z, 7)])              # multiple substitution

ex.evalf(subs={x:3, y:4, z:sp.pi})             # numerical evaluation
ex.evalf(100, subs={x:3, y:4, z:sp.pi})        # 100 digits of accuracy

ex = (x**2 + 3*x + 2)/(x+1)
ex.simplify()                                  # simplify: three versions
sp.simplify(ex)
sp.simplify( (x**2 + 3*x + 2)/(x+1) )

ex = (x+2)*(x-3)                               # expand : three versions
ex.expand()
sp.expand(ex)
sp.expand( (x+2)*(x-3) )

ex = x**3 - x**2 + x - 1
ex.factor()
sp.factor(ex)
sp.factor( x**3 - x**2 + x -1 )

# collect, cancel, apart, trigsimp, expand_trig, powsimp
# expand_power_exp, expand_power_base, pow_denest, expand_log, etc.

ex = sp.exp(x*y*z)
ex.diff(x)                    # differentiate: three versions
sp.diff(ex,x)
sp.diff(sp.exp(x**2), x)
sp.diff(ex, x,x,x)            # 3rd derivative
ex.diff(x,y,z)                # 3rd mixed derivative
```

```
exD = sp.Derivative(ex, x,y,z)  # expression with derivative
exD.doit()                      # evaluate the derivative expression

ex = x**2 + y**2
ex.integrate(x)                       # integrate: three versions
sp.integrate(ex, x)                   # (note, the constant is left off)
sp.integrate(x**2, x)
sp.integrate(ex, (x,0,sp.oo))         # definite integral
sp.integrate(ex, (x,1,2), (y, 0, z)) # double integral

exI = sp.Integral(ex, x)              # expression with integral
exI = sp.Integral(ex, (x,0,2))        # include bounds
exI.doit()                            # evaluate the integral expression

ex = sp.sin(x)/x
exL = sp.limit(ex, x, 0)              # limit
exL = sp.Limit(ex, x, 0)              # expression with limit
exL.doit()                            # evaluate the limit expression

sp.solve(x**2-y, x)                   # solve
ex = sp.Eq(x**2, y)                   # create an equality: x**2 = y
sp.solve(ex, x)
sp.solve( (x-y+2, x+y-3), (x, y) )    # solve system of equations
sp.solve( (x*y-7, x+y-6), (x, y) )    # nonlinear: try replace 6 with z

f = sp.Function('f')                  # function variable: 2 ways
f = sp.symbols('f', cls=sp.Function)

diffEq = f(x).diff(x,x) - 2*f(x).diff(x) + f(x) - sp.sin(x)
sp.dsolve(diffEq, f(x))               # solve ODE for f(x)

m11, m12, m21, m22 = sp.symbols("m11, m12, m21, m22")
b1, b2             = sp.symbols("b1, b2")
A = sp.Matrix([ [m11, m12], [m21, m22] ])
b = sp.Matrix([b1,b2])
A.inv()*b
A**-1 * b
```

```
symbols
Function

subs

evalf

simplify

expand
factor

diff
integrate
limit
Derivative
Integral
Limit
doit

solve
dsolve

Matrix
row
row_del
inv
```

# Class 26: Widgets

- Rate equations
- Symbolic math
- Widgets
- Python-Excel interface

```
import ipywidgets as wgt
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

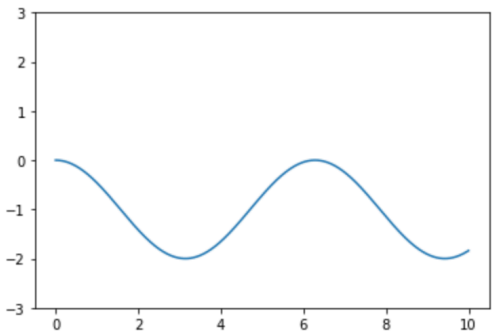
```
def f(a,b,c):
    x = np.linspace(0,10,1000)
    y = np.cos(a*x + b) + c
    plt.plot(x,y)
    plt.ylim([-3,3])
    plt.show()

wgt.interact(f, a=(-5,5), b=(-5,5), c=(-6,6));
```

a 1

b 0

c -1




```
import ipywidgets as wg
from IPython.display import display
import numpy as np
```

```
Title = wg.HTML(value="<br><b>Enter properties:</b>")
display(Title)

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

tab = wg.Label('',               layout=wg.Layout(width='3%'))
L0  = wg.Label('Some quantity:', layout=wg.Layout(width='30%'))
R0  = wg.Text(value='100',       layout=wg.Layout(width='20%'))

box0 = wg.HBox([tab, L0, R0])
display(box0)

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

L1 = wg.Label('Some radio buttons:',               layout=wg.Layout(width='30%'))
R1 = wg.RadioButtons(options=['dog', 'cat', 'pig', 'cow', 'snake'], value='dog',
                                                   layout=wg.Layout(width='20%'))

box1 = wg.HBox([tab, L1, R1])
display(box1)

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

L2 = wg.Label('Dropdown box:', layout=wg.Layout(width='30%'))
R2 = wg.Dropdown(options=['BYU','USU','Utah','UVU'], value='BYU',
                         layout=wg.Layout(width='30%'),
                         description='', button_style='')

box2 = wg.HBox([tab, L2, R2])
display(box2)

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

submitButton = wg.Button(description='Submit and Run', button_style='success')

display(submitButton)

def buttonClicked(sbutton):
    res0.value = "The value of R0 is " + str(R0.value)
    res1.value = "The value of R1 is " + str(R1.value)
    res2.value = "The value of R2 is " + str(R2.value)

submitButton.on_click(buttonClicked)

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

Results = wg.HTML(value="<br><b>Results:</b>")
display(Results)

res0 = wg.Label(layout=wg.Layout(width='25%'))
res1 = wg.Label(layout=wg.Layout(width='25%'))
res2 = wg.Label(layout=wg.Layout(width='25%'))

display(res0, res1, res2)
```

**Enter properties:**

Some quantity: 100

Some radio buttons:
- (•) dog
- ( ) cat
- ( ) pig
- ( ) cow
- ( ) snake

Dropdown box: BYU

Submit and Run

**Results:**

# Class 27: Excel Interface: xlwings


- Excel macros
  - Name
  - Keyboard shortcut
  - Relative references
  - Can edit code in vba; can use to see code corresponding to operation
- Enable Developer:
  - Windows: File → options → Ribbon → Developer
  - Mac: Excel → Preferences → Ribbon → Developer
- Interact with Excel from Python
- Interact with Python from Excel
  - Setup: run terminal commands: xlwings runtime install, xlwings addin install
  - Set the PYTHONPATH nad the python command location in the xlwings tab in Excel
  - In terminal: xlwings quickstart project_name
  - Creates folder project_name with files project_name.py and project_name.xlsm

```
#-------- open
wb  = xw.Book('data.xlsx')
sht = wb.sheets['Sheet1']

#-------- put/get variables
var1 = 2.20462
sht.range('A6').value = var1
var2 = sht.range('A7').value

#-------- put/get arrays
t = sht.range('E2:E10').value
t = sht.range('E2:E10').options(np.array).value
t = sht.range('E2').expand('vertical').value

sht.range('F2').value = t[:, np.newaxis]

#-------- formula
sht.range('H2').formula = "=sum(F:F)"

#-------- sheets
wb.sheets.add("new sheet", after="sheet 1")
print(wb.sheets)

#-------- save/close
wb.save()
wb.close()
```

# Debugging

**Key types of bugs (among many):**

- **syntax errors**
  - Improper python
  - Missing ":" or indentation, or spelling, or invalid expressions.
  - Usually easily found. Python will tell you about them.
  - Often parentheses issues.
- **runtime errors**
  - An error that occurs when you run the code.
  - Divide by zero, or an improper value.
- **logical errors**
  - You coded it wrong.
  - But the computer doesn't know that. It tries to solve the problem.
  - You might get an answer but it might be wrong.
  - You might not get an answer. Like fsolve cannot converge...

## Others

- Exceeding array bounds, or not being careful about indexing.
- Using `^` instead of `**`
- Using `6.02*10**23` instead of 6.02E23 (more of a computational sin than a bug).
- Using conflicting names; like using `f` for both a function and a variable.
- Passing a function *call* as an argument instead of passing the function itself as the argument:
  - Like using this: `I = quad(f(x), a, b)[0]`
  - instead of this: `I = quad(f, a, b)[0]`

## The Π Rule

- **"However long you think it should take, multiply by Π."**
- Bugs are the reason for the Π rule!
- Time t spend being careful and avoiding bugs saves you Πt debugging!