Shall We Program?

Python For Analytics

What are you expected to know?

- Do I need to be a programming expert?

- Do I need to be an IT expert?

- Do I need to know Python?

- Do I need to face a Python?

Computers Want to be Helpful…

- Computers are built for one purpose - to do things for us

- But we need to speak their language to describe what we want done

- Users have it easy - someone already put many different programs (instructions) into the computer and users just pick the ones they want to use
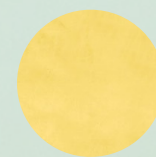
What Next?

What Next?
What Next?
What Next?

What Next?
What Next?
What Next?

Users vs. Programmers

- Users see computers as a set of tools - word processor, spreadsheet, map, to-do list, etc.

- Programmers learn the computer "ways" and the computer language

- Programmers have some tools that allow them to build new tools

- Programmers sometimes write tools for lots of users and sometimes programmers write little "helpers" for themselves to automate a task

Why be a Programmer?

- To get some task done - we are the user and programmer

  - Clean up survey data

- To produce something for others to use - a programming job

  - Fix a performance problem in the Sakai software

  - Add a guestbook to a web site

Can you count?

the clown ran after the car and the car ran into the tent
and the tent fell down on the clown and the car

How about an entire file, can you count?
Within Million words?

```
name = input('Enter file:')
handle = open(name)

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1
```

Early Learner: Syntax Errors

- We need to learn the Python language so we can communicate our instructions to Python. In the beginning we will make lots of mistakes and speak gibberish like small children.

- When you make a mistake, the computer does not think you are "cute". It says "syntax error" - given that it knows the language and you are just learning it. It seems like Python is cruel and unfeeling.

- You must remember that you are intelligent and can learn. The computer is simple and very fast, but cannot learn. So it is easier for you to learn Python than for the computer to learn English…

Talking to Python

csev$ python3

Python 3.5.1 (v3.5.1:37a07cee5969, Dec 5 2015, 21:12:44) [GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwinType "help", "copyright", "credits" or "license" for more information.
>>>

What's next?

```
csev$ python3
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  5 2015,
21:12:44) [GCC 4.2.1 (Apple Inc. build 5666) (dot
3)] on darwinType "help", "copyright", "credits" or
"license" for more information.
>>> x = 1
>>> print(x)
1
>>> x = x + 1
>>> print(x)
2
>>> exit()
```

This is a good test to make sure that you have Python correctly installed. Note that quit() also works to end the interactive session.

Elements of Python

- **Vocabulary / Words** - Variables and Reserved words (Chapter 2)

- **Sentence structure** - valid syntax patterns (Chapters 3-5)

- **Story structure** - constructing a program for a purpose

Reserved Words

You cannot use reserved words as variable names / identifiers

False	class	return	is	finally
None	if	for	lambda	continue
True	def	from	while	nonlocal
and	del	global	not	with
as	elif	try	or	yield
assert	else	import	pass	
break	except	in	raise	

Sentences or Lines

```
x = 2        ⟵    Assignment statement
x = x +2     ⟵    Assignment with expression
print(x)     ⟵    Print statement
```

Variable   Operator   Constant   Function

Programming Paragraphs

Python Scripts

- Interactive Python is good for experiments and programs of 3-4 lines long.

- Most programs are much longer, so we type them into a file and tell Python to run the commands in the file.

- In a sense, we are "giving Python a script".

- As a convention, we add ".py" as the suffix on the end of these files to indicate they contain Python.

Interactive versus Script

- Interactive

  - You type directly to Python one line at a time and it responds

- Script

  - You enter a sequence of statements (lines) into a file using a text editor and tell Python to execute the statements in the file

  - Some steps are conditional - they may be skipped.

  - Sometimes a step or group of steps is to be repeated.

  - Sometimes we store a set of steps to be used over and over as needed several places throughout the program

Let's run out first script

```
name = input('Enter file:')
handle = open(name, 'r')

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] =
counts.get(word,0) + 1

bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count >
bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
```

Sequential

Repeated

Conditional

```python
name = input('Enter file:')
handle = open(name, 'r')

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] =
counts.get(word,0) + 1

bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count >
bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
```

A short Python "Story" about how to count words in a file

A word used to read data from a user

A sentence about updating one of the many counts

A paragraph about how to find the largest item in a list

Constants

- Fixed values such as numbers, letters, and strings, are called "constants" because their value does not change

- Numeric constants are as you expect

- String constants use single quotes (')
or double quotes (")

```
>>> print(123)
123
>>> print(98.6)
98.6
>>> print('Hello world')
Hello world
```

Variables

- A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable "name"

- Programmers get to choose the names of the variables

- You can change the contents of a variable in a later statement

```
x = 12.2
y = 14
```

x | 12.2

y | 14

Variables

- A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable "name"

- Programmers get to choose the names of the variables

- You can change the contents of a variable in a later statement

```
x = 12.2
y = 14
x = 100
```

x | 12.2 100

y | 14

Python Variable Name Rules

- Must start with a letter or underscore _

- Must consist of letters, numbers, and underscores

- Case Sensitive

```
Good:       spam    eggs    spam23    _speed
Bad:        23spam      #sign   var.12
Different:      spam    Spam    SPAM
```

Mnemonic Variable Names

- Since we programmers are given a choice in how we choose our variable names, there is a bit of "best practice"

- We name variables to help us remember what we intend to store in them ("mnemonic" = "memory aid")

- This can confuse beginning students because well-named variables often "sound" so good that they must be keywords

http://en.wikipedia.org/wiki/Mnemonic

```
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * 1q3z9afd
print(x1q3p9afd)
```

What is this
bit of code
doing?

```
x1q3z9ocd = 35.0                        a = 35.0
x1q3z9afd = 12.50                       b = 12.50
x1q3p9afd = x1q3z9ocd * 1q3z9afd        c = a * b
print(x1q3p9afd)                        print(c)
```

What are
these bits of
code doing?

```
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * 1q3z9afd
print(x1q3p9afd)
```

```
a = 35.0
b = 12.50
c = a * b
print(c)
```

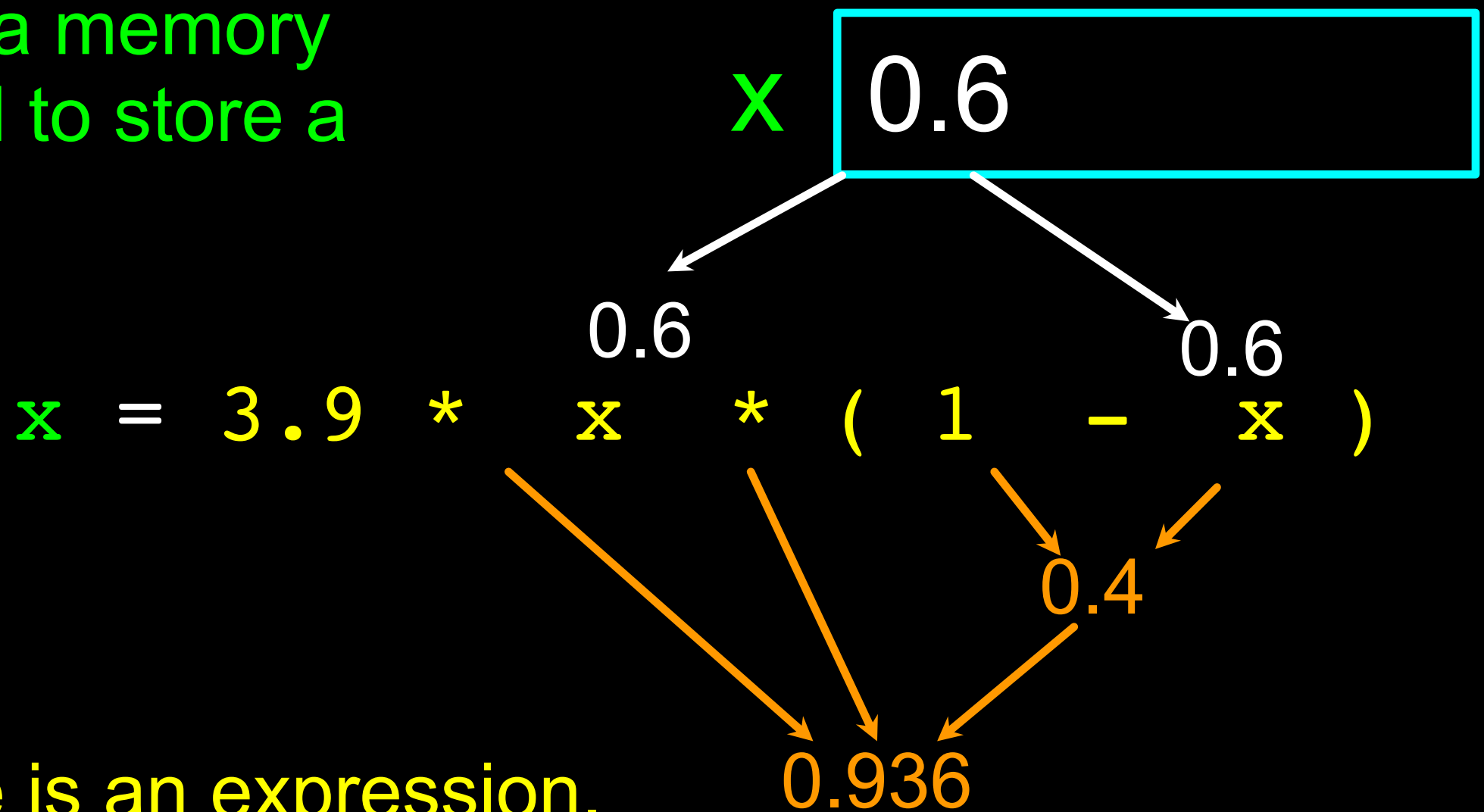What are
these bits of
code doing?

```
hours = 35.0
rate = 12.50
pay = hours * rate
print(pay)
```

Assignment Statements

- We assign a value to a variable using the assignment statement (=)

- An assignment statement consists of an expression on the right-hand side and a variable to store the result

```
x = 3.9 * x * ( 1 - x )
```
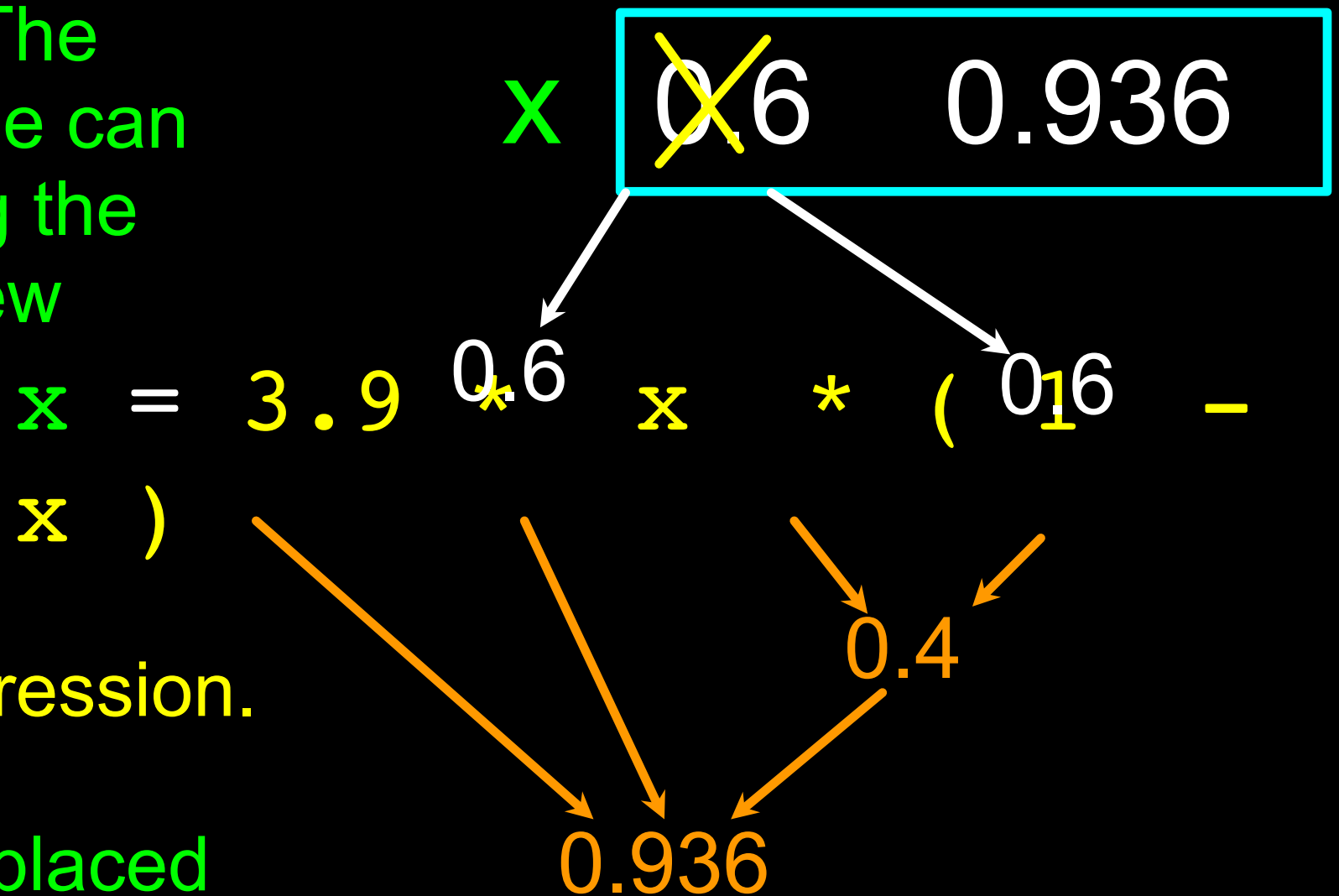
A variable is a memory
location used to store a
value (0.6)

x 0.6

0.6   0.6

x = 3.9 * x * (1 - x)

0.4

0.936

The right side is an expression.
Once the expression is evaluated,
the result is placed in (assigned to) x.

A variable is a memory location used to store a value. The value stored in a variable can be updated by replacing the old value (0.6) with a new value (0.936).

x = 3.9 * x * (1 - x)

The right side is an expression. Once the expression is evaluated, the result is placed in (assigned to) the variable on the left side (i.e., x).

x   0.6   0.936

0.6   0.6

1

0.4

0.936

Expressions…

Numeric Expressions

- Because of the lack of mathematical symbols on computer keyboards - we use "computer-speak" to express the classic math operations

- Asterisk is multiplication

- Exponentiation (raise to a power) looks different than in math

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

Numeric Expressions

```
>>> xx = 2
>>> xx = xx + 2
>>> print(xx)
4
>>> yy = 440 * 12
>>> print(yy)
5280
>>> zz = yy / 1000
>>> print(zz)
5.28
```

```
>>> jj = 23
>>> kk = jj % 5
>>> print(kk)
3
>>> print(4 ** 3)
64
```

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

Order of Evaluation

- When we string operators together - Python must know which one to do first

- This is called "operator precedence"

- Which operator "takes precedence" over the others?

$$x = 1 + 2 * 3 - 4 /$$
$$5 ** 6$$

Operator Precedence Rules

Highest precedence rule to lowest precedence rule: Parenthesis

- Parentheses are always respected

Power

- Exponentiation (raise to a power)

Multiplication

- Multiplication, Division, and Remainder

Addition

- Addition and Subtraction

- Left to right

Left to Right

```
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print(x)
11.0
>>>
```

Parenthesis
Power
Multiplication
Addition
Left to Right

```
1 + 2 ** 3 / 4 * 5

1 + 8 / 4 * 5

1 + 2 * 5

1 + 10

11
```

Operator Precedence

Parenthesis
Power
Multiplication
Addition
Left to Right

- Remember the rules top to bottom

- When writing code - use parentheses

- When writing code - keep mathematical expressions simple enough that they are easy to understand

- Break long series of mathematical operations up to make them more clear

What Does "Type" Mean?

- In Python variables, literals, and constants have a "type"

- Python knows the difference between an integer number and a string

- For example "+" means "addition" if something is a number and "concatenate" if something is a string

```
>>> ddd = 1 + 4
>>> print(ddd)
5
>>> eee = 'hello ' + 'there
>>> print(eee)
hello there
```

concatenate = put together

Type Matters

- Python knows what "type" everything is

- Some operations are prohibited

- You cannot "add 1" to a string

- We can ask Python what type something is by using the type() function

```
>>> eee = 'hello ' + 'there'
>>> eee = eee + 1
Traceback (most recent call last):  File "<stdin>", line 1, in <module>TypeError: Can't convert 'int' object to str implicitly
>>> type(eee)
<class'str'>
>>> type('hello')
<class'str'>
>>> type(1)
<class'int'>
>>>
```

Several Types of Numbers

- Numbers have two main types

  - Integers are whole numbers: -14, -2, 0, 1, 100, 401233

  - Floating Point Numbers have decimal parts: -2.5 , 0.0, 98.6, 14.0

- There are other number types - they are variations on float and integer

```
>>> xx = 1
>>> type (xx)
<class 'int'>
>>> temp = 98.6
>>> type(temp)
<class'float'>
>>> type(1)
<class 'int'>
>>> type(1.0)
<class'float'>
>>>
```

Type Conversions

- When you put an integer and floating point in an expression, the integer is implicitly converted to a float

- You can control this with the built-in functions int() and float()

```
>>> print(float(99) + 100)
199.0
>>> i = 42
>>> type(i)
<class'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class'float'>
>>>
```

Integer Division

Integer division produces a floating point result

This was different in Python 2.x

```
>>> print(10 / 2)
5.0
>>> print(9 / 2)
4.5
>>> print(99 / 100)
0.99
>>> print(10.0 / 2.0)
5.0
>>> print(99.0 / 100.0)
0.99
```

String Conversions

- You can also use int() and float() to convert between strings and integers

- You will get an error if the string does not contain numeric characters

```
>>> sval = '123'
>>> type(sval)
<class 'str'>
>>> print(sval + 1)
Traceback (most recent call last):  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
>>> ival = int(sval)
>>> type(ival)
<class 'int'>
>>> print(ival + 1)
124
>>> nsv = 'hello bob'
>>> niv = int(nsv)
Traceback (most recent call last):  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'x'
```

User Input

- We can instruct Python to pause and read data from the user using the input() function

- The input() function returns a string

```
nam = input('Who are you? ')
print('Welcome', nam)
```

Who are you?
Chuck
Welcome Chuck

Converting User Input

- If we want to read a number from the user, we must convert it from a string to a number using a type conversion function

- Later we will deal with bad input data

```
inp = input('Europe floor?')
usf = int(inp) + 1
print('US floor', usf)
```

Europe floor? 0
US floor 1

Comments in Python

- Anything after a # is ignored by Python

- Why comment?

    - Describe what is going to happen in a sequence of code

    - Document who wrote the code or other ancillary information

    - Turn off a line of code - perhaps temporarily

```python
# Get the name of the file and open it
name = input('Enter file:')
handle = open(name, 'r')

# Count word frequency
counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1

# Find the most common word
bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

# All done
print(bigword, bigcount)
```

Summary

- Type

- Reserved words

- Variables (mnemonic)

- Operators

- Operator precedence

- Integer Division

- Conversion between types

- User input

- Comments (#)

Exercise

Write a program to prompt the user for hours and rate per hour to compute gross pay.

```
Enter Hours: 35
Enter Rate: 2.75

Pay: 96.25
```

Let's do it all again but now on jupyter notebook