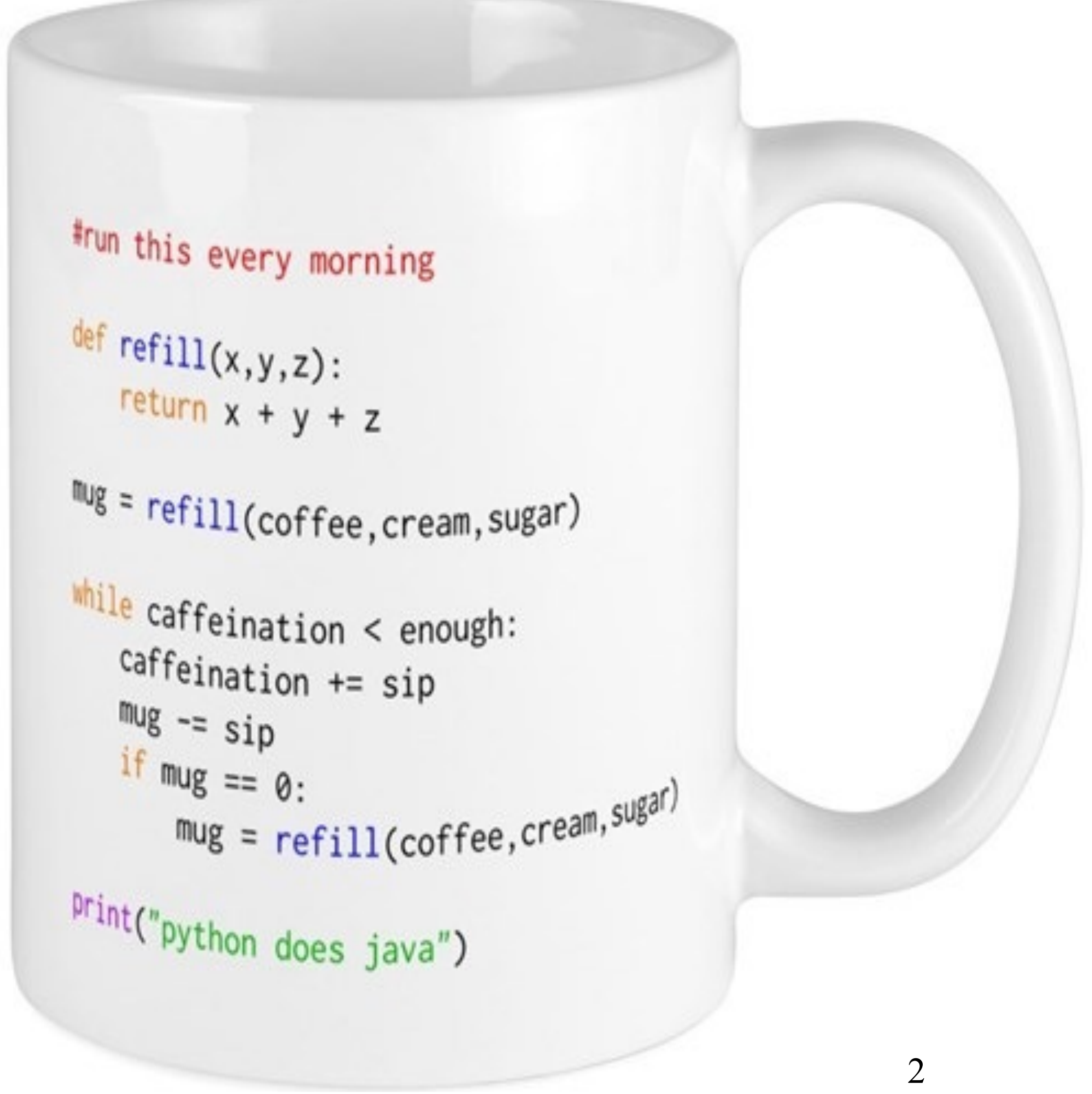
Objectives for class 2

--- Chapter 1---
1.5 To use sound programming style and document programs properly (§1.7).
1.6 To explain the differences between syntax errors, runtime errors, and logic errors (§1.8).

--- Chapter 2---
2.1 To write programs that perform simple computations (§2.2)
2.2 To obtain input from a program's user by using the input function and to convert strings to numbers using the int and float functions (§2.3)
2.3 To use identifiers to name elements such as variables and functions (§2.4)
2.4 To assign data to variables (§2.5)
2.5 To define named constants (§2.7)
2.6 To use the operators +, −, *, /, //, %, and ** (§2.8)
2.7 To program using division and remainder operators (§2.9)
2.8 To write and evaluate numeric expressions (§2.10)

Anatomy of a Python Program

- Statements
- Comments
- Indentation



```
#run this every morning

def refill(x,y,z):
    return x + y + z

mug = refill(coffee,cream,sugar)

while caffeination < enough:
    caffeination += sip
    mug -= sip
    if mug == 0:
        mug = refill(coffee,cream,sugar)

print("python does java")
```

Statements – Represents Action(s)

- A statement represents an action or actions.
  - print(“python does java")
  - Action: displays the greeting "Welcome to Python“.

Statements

```
#run this every morning
```

```
def refill(x,y,z):
    return x + y + z

mug = refill(coffee, cream, sugar)

while caffeination < enough :
    caffeination += sip
    mug -= sip
    if mug == 0 :
        mug = refill(coffee, cream, mug)

print("python does java")
```

Comments – Ignored by the Python

- Anything after a **#**
- Why comment?
  - Describe what will happen in a code segment
  - Document who wrote the code or other ancillary information
  - Turn off a line of code (perhaps temporarily)

Comments

```
#run this every morning

def refill(x,y,z):
    return x + y + z

mug = refill(coffee, cream, sugar)

while caffeination < enough :
    caffeination += sip
    mug -= sip
    if mug == 0 :
        mug = refill(coffee, cream, mug)

print("python does java")
```

Indentation - Indicate a Block of Code

- The spaces at the beginning of a code line.
- Matters in Python.
- First line has no indentation
- Use the same number of spaces in the same block of code, otherwise wrong.

Wrong indentation causes an error

```
#Display two messages
    print("Welcome to Python")
print("Python is fun")
```

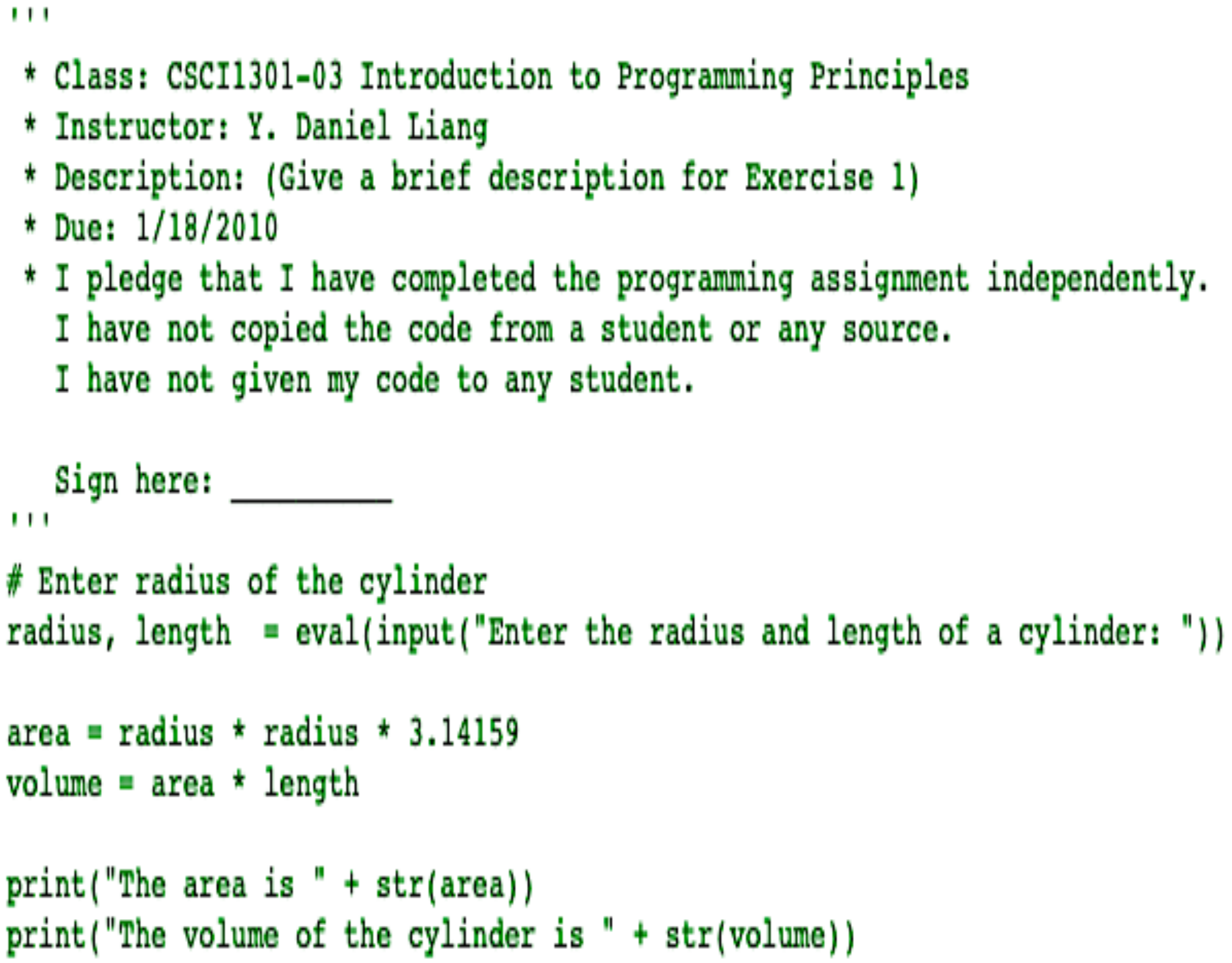
Indentation

Indentation

```
#run this every morning

def refill(x,y,z):
    return x + y + z

mug = refill(coffee, cream, sugar)

while caffeination < enough :
    caffeination += sip
    mug -= sip
    if mug == 0 :
        mug = refill(coffee, cream, mug)

print("python does java")
```

How to write good programs?

- Appropriate Comments
  - Authors
  - Key features
  - unique techniques
- Proper Indentation and Spacing Lines
  - Indent four spaces
  - blank line to separate segments of the code.



```
'''
 * Class: CSCI1301-03 Introduction to Programming Principles
 * Instructor: Y. Daniel Liang
 * Description: (Give a brief description for Exercise 1)
 * Due: 1/18/2010
 * I pledge that I have completed the programming assignment independently.
   I have not copied the code from a student or any source.
   I have not given my code to any student.

   Sign here: ________
'''

# Enter radius of the cylinder
radius, length  = eval(input("Enter the radius and length of a cylinder: "))

area = radius * radius * 3.14159
volume = area * length

print("The area is " + str(area))
print("The volume of the cylinder is " + str(volume))
```

https://liangpy.pearsoncmg.com/supplement/codingguidelines.html

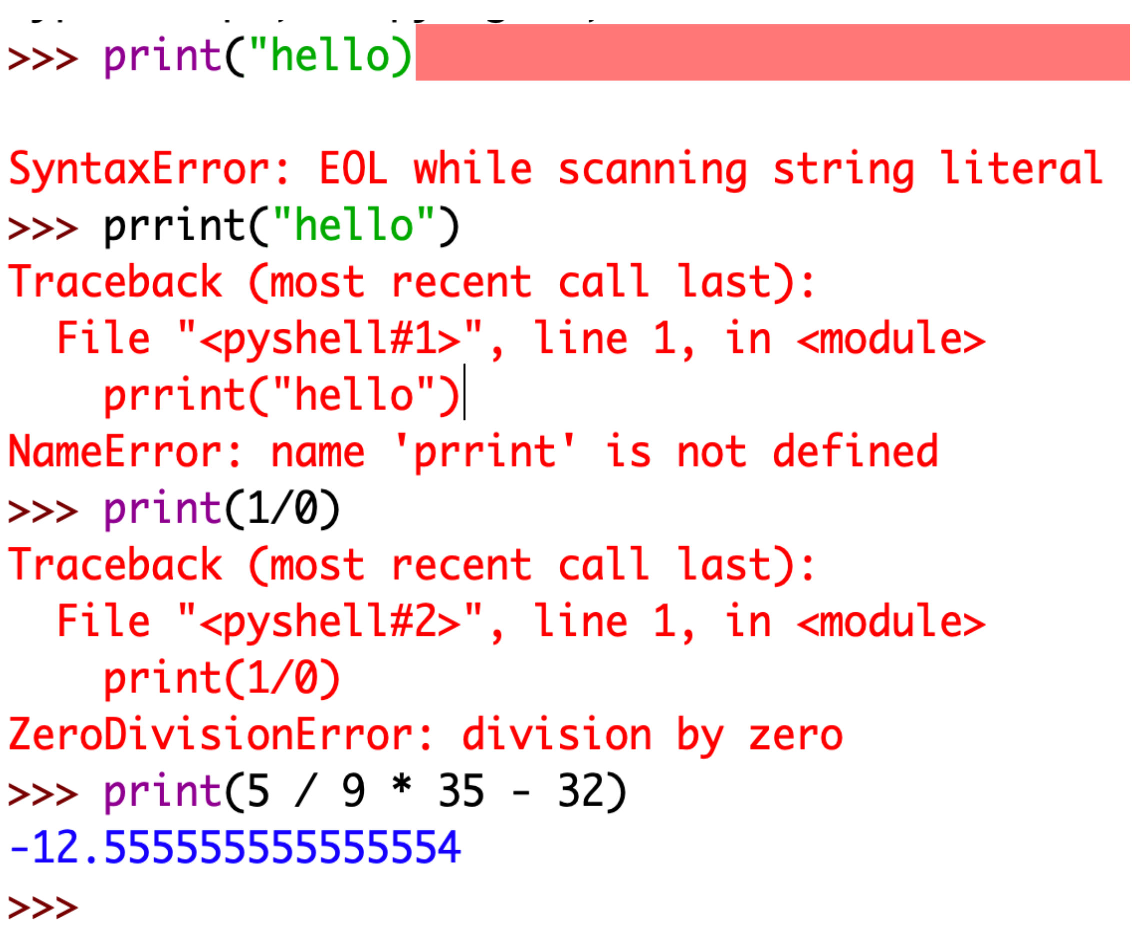
Three Types of Errors In Programs

- Syntax Errors
  - Error in code construction

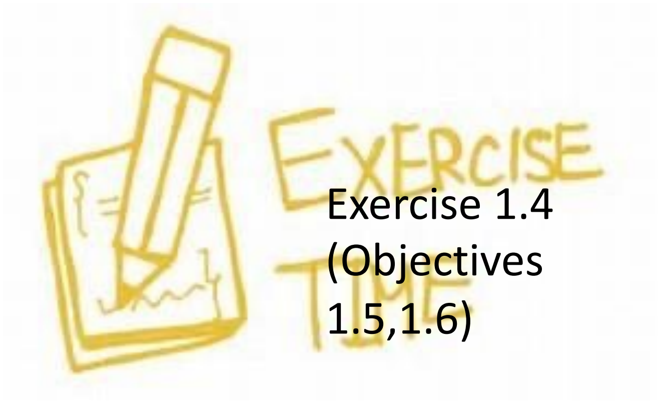
    Can not RUN / Not executable
- Runtime Errors
  - Causes the program to abort

    Able to run, but quits while running
- Logic Errors
  - Produces incorrect result

    Able to run, but results not expected



```
>>> print("hello)

SyntaxError: EOL while scanning string literal
>>> prrint("hello")
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    prrint("hello")
NameError: name 'prrint' is not defined
>>> print(1/0)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    print(1/0)
ZeroDivisionError: division by zero
>>> print(5 / 9 * 35 - 32)
-12.555555555555554
>>>
```

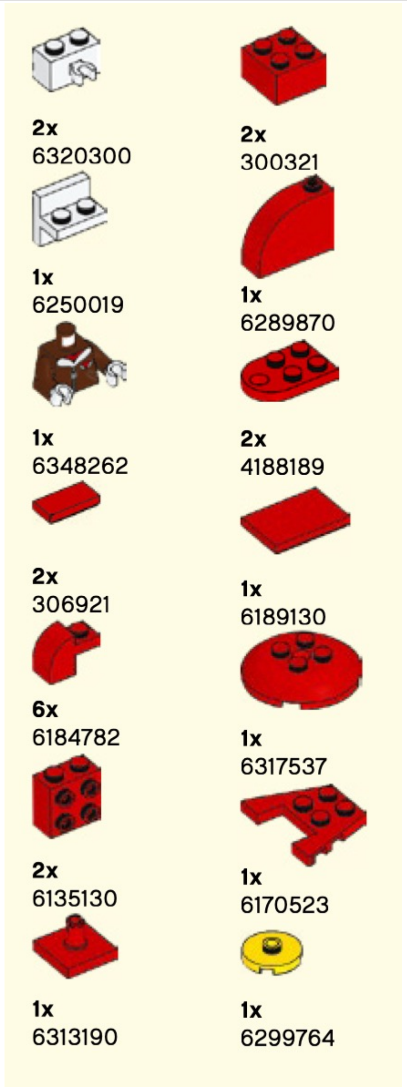


Exercise 1.4 (Objectives 1.5,1.6)

Writing A Program Is Like Building A LEGO

F

LEGO



2x	6320300	2x	300321
1x	6250019	1x	6289870
1x	6348262	2x	4188189
2x	306921	1x	6189130
6x	6184782	1x	6317537
2x	6135130	1x	6170523
1x	6313190	1x	6299764

Different bricks

Constants- the Easiest brick

```
#Enter radius of cylinder
radius,length = eval(input("Enter the radius and length of cylinder:"))

area=radius * radius * 3.14159
volume=area * length

print("The area is" + str(area))
print("The volume of cylinder is " + str(volume))
```

Constants- the easiest brick

- Fixed values
- Numeric constants are as you expect
- String constants use single quotes (‘) or double quotes (")

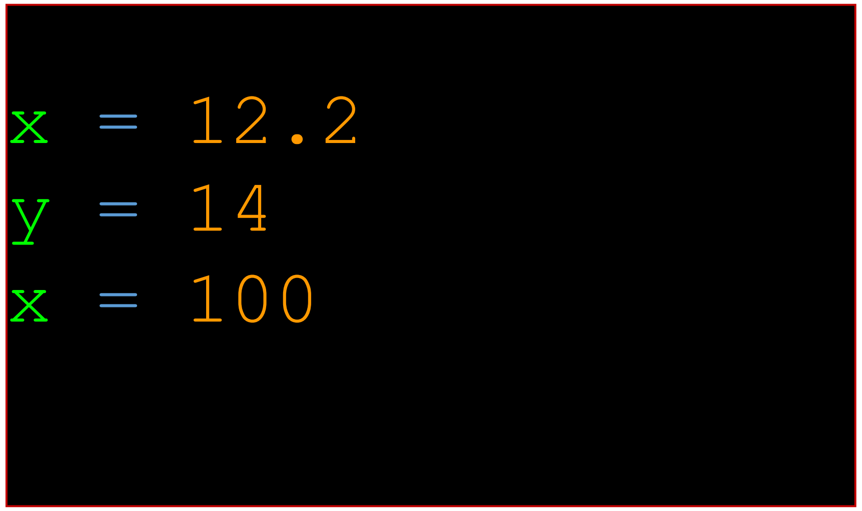
```
>>> print(123)
123
>>> print(98.6)
98.6
>>> print('Hello world')
Hello world
```

Variables – Most Commonly used brick

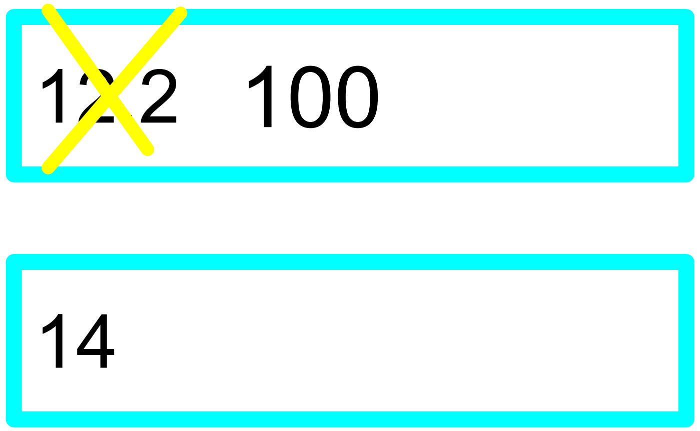
```
#Enter radius of cylinder
radius,length = eval(input("Enter the radius and length of cylinder:"))

area = radius * radius * 3.14159
volume=area * length

print("The area is" + str(area))
print("The volume of cylinder is " + str(volume))
```

Variables – Most Commonly used brick

- A name that refers to a value.
- Programmers choose the names.
- The value referred to can be modified



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



Python Variable Name Rules

- Must start with a letter or underscore _
- Must consist of letters, numbers, and underscores
- Case Sensitive
- Cannot be a **reserved word**

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

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
```

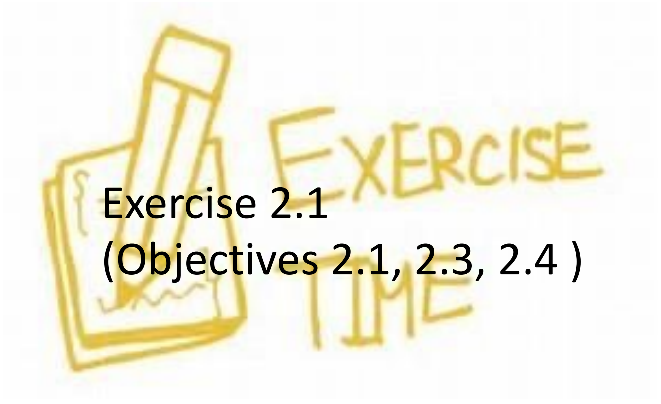
Mnemonic Variable Names

- "mnemonic" = "memory aid"
- Help us remember what we intend to store

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

Are they written for the same problem?

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



Exercise 2.1
(Objectives 2.1, 2.3, 2.4)

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

What statements we have learned so far?

```
x = 2
x = x + 2
print(x)
```

Assignment statement
Assignment with expression
Print statement

Variable Operator Constant Function

Assignment Statements – Bricks assign values to variables

```
#Enter radius of cylinder
radius,length = eval(input("Enter the radius and length of cylinder:"))

area = radius * radius * 3.14159
volume=area * length

print("The area is" + str(area))
print("The volume of cylinder is " + str(volume))
```

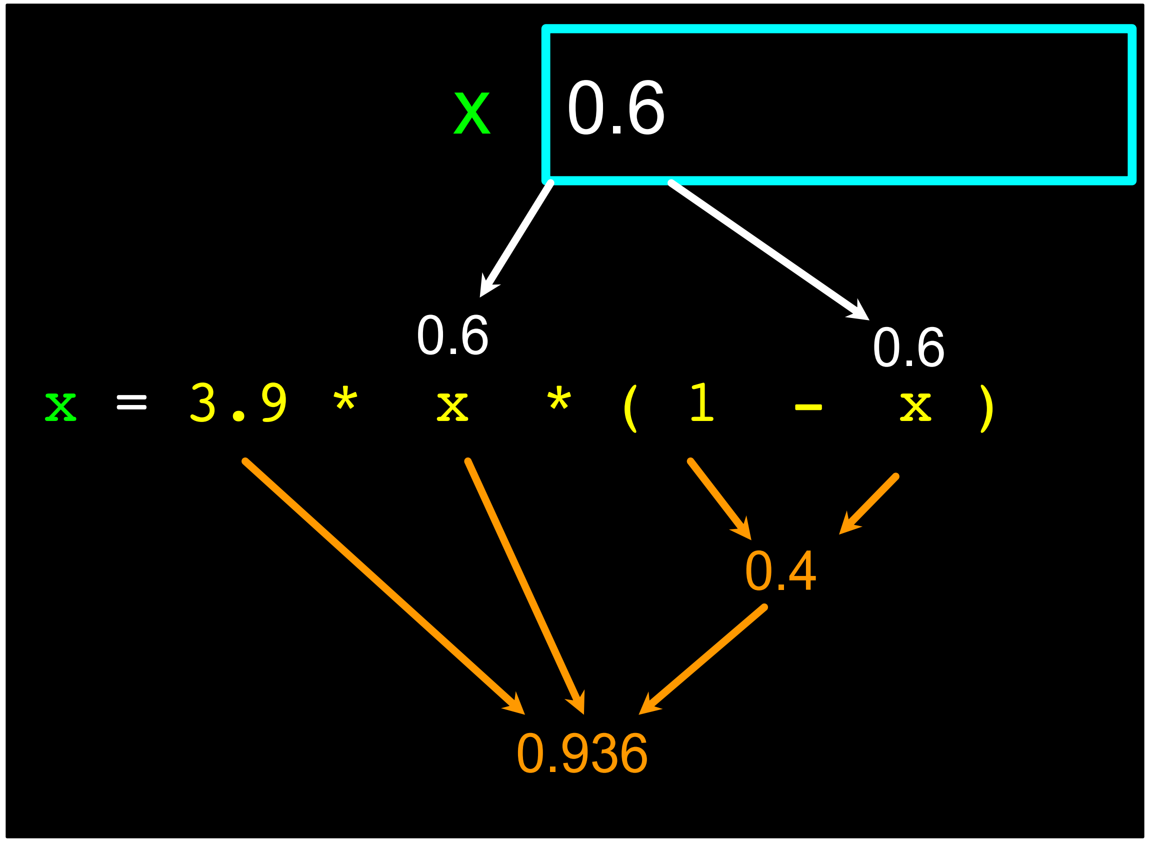
Assignment Statements – Bricks assign values to variables

- Create a variable or change a value referred to a variable
- 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)

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



Expressions – Bricks evaluate to a value

- How to write an expression?
  - A combination of values, variables, and operators.
  - A value itself is an expression, and so is a variable

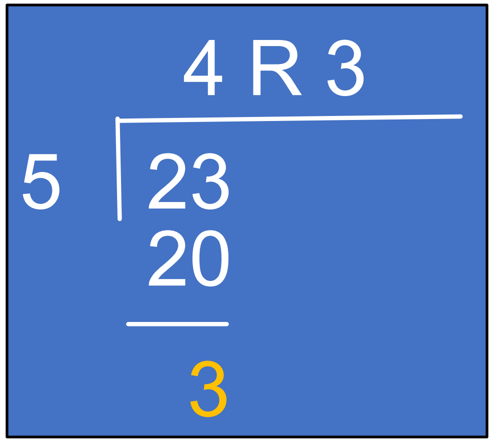
- A single line expression is special
  - In interactive mode, the interpreter evaluates it and displays the result.
  - In script mode, the interpreter does not display the result

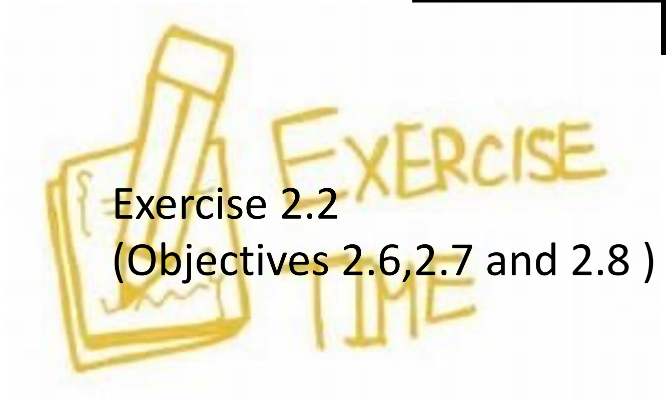
Numeric Expressions – evaluate to ints or floats

```
>>> 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





Exercise 2.2
(Objectives 2.6,2.7 and 2.8)

Input Statements – Bricks obtaining input from user

- 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? Yuan
Welcome Yuan
```

Converting User Input to numbers

- To read a number from the user, convert it from a string to a number using a type conversion function.

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

```
Europe floor? 0
US floor 1
```

Order of Evaluation in Expression

- When we string operators together - Python must know which one to do first
- Called "operator precedence"
- Which operator "takes precedence" over the others?

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

Operator Precedence Rules – decide the evaluation order

- Highest precedence rule to lowest precedence rule:
  - Parentheses are always respected
  - Exponentiation (raise to a power)
  - Multiplication, Division, and Remainder
  - Addition and Subtraction

- What if we have operators with same precedence?
  - Left to right

()

**

* / %

+

Left to Right

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

()

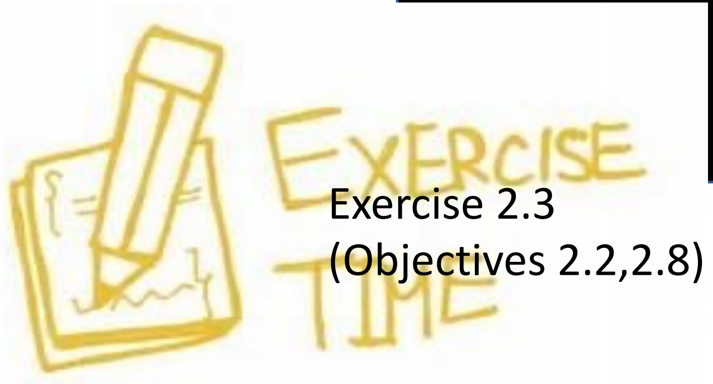
**

* / %

+

Left to Right

$$\begin{tikzcd} \text{1 + 2 ** 3 / 4 * 5} \arrow[d] \\ \text{1 + 8 / 4 * 5} \arrow[d] \\ \text{1 + 2 * 5} \arrow[d] \\ \text{1 + 10} \arrow[d] \\ \text{11} \end{tikzcd}$$



Exercise 2.3
(Objectives 2.2,2.8)

Summary

- Constants
- Reserved Words
- Variables
    - Name Rules
- Assignment Statement
- Expressions
    - Numeric Expressions
    - Order of Evaluation
    - Operator Precedence Rule
- Input Statement