MAT 331 Python Install… …….

INSTALLING PYTHON

- We shall be using Python 3.5 (and above) in this course. (Provided by Virtual SINC Site)
- For people who have installed Python+Jupyter or Anaconda: Create a file and run the following code:

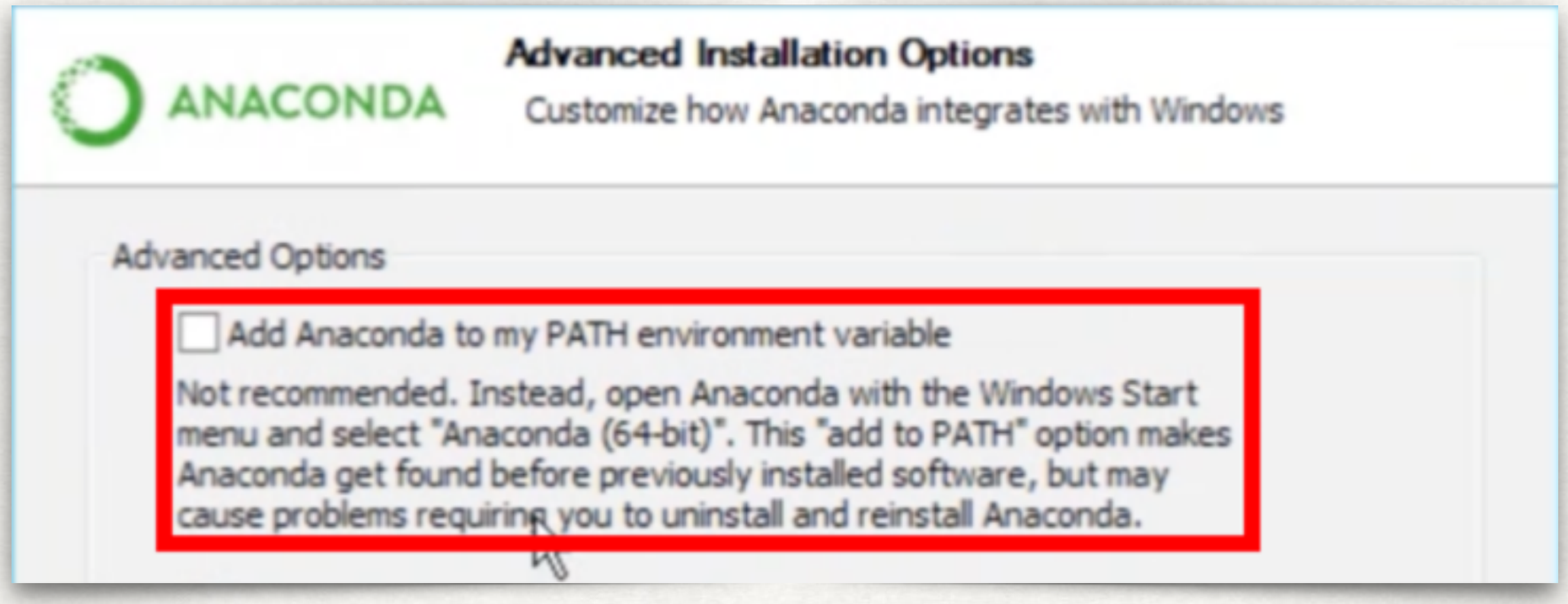
```
import sys
print (sys.version)
```

- **If you have installed the Python 3.5 (or above) and Jupyter you are done for now**

INSTALLATION

- Download Anaconda from here: https://www.continuum.io/downloads
- There are two ways to do this. Choose the 3.6 version if you have no Python installed at all.
- If you installed The 2.7 version
- You have some work to do.

WINDOWS


ANACONDA
Advanced Installation Options
Customize how Anaconda integrates with Windows
Advanced Options
Add Anaconda to my PATH environment variable
Not recommended. Instead, open Anaconda with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.

FOR PEOPLE WHO INSTALLED 2.7

- Create a conda environment for python 3.5 with anaconda:
- conda create -n py36 python=3.6 anaconda

- Update it
- conda update -n py36 anaconda
- This will pull down all the 3.6 stuff.

WINDOWS

If you don't know where your conda and/or python is, you type the following commands into your anaconda prompt

```
(C:\Users\mgalarnyk\Anaconda2) C:\Users\mgalarnyk>where python
C:\Users\mgalarnyk\Anaconda2\python.exe

(C:\Users\mgalarnyk\Anaconda2) C:\Users\mgalarnyk>where conda
C:\Users\mgalarnyk\Anaconda2\Scripts\conda.exe
```

Next, you can add **Python and Conda** to your path by using the setx command in your command prompt.

```
C:\Users\mgalarnyk>SETX PATH "%PATH%;C:\Users\mgalarnyk\Anaconda2\Scripts;C:\Use
rs\mgalarnyk\Anaconda2"

SUCCESS: Specified value was saved.
```

RUNNING IN A CONDA ENVIRONMENT

- You should “activate it”
- source activate py36
- (or, if you have anaconda on a non-standard place eg:
- source /anaconda/envs/py36/bin/activate py36)
- You ought to read more about the conda command. See http://conda.pydata.org/docs/_downloads/conda-cheatsheet.pdf (critical) and https://jakevdp.github.io/blog/2016/08/25/conda-myths-and-misconceptions/ (to understand the difference between conda and pip and why python has both (we’ll use both in this course).

MAT 331 Python Variables, Expressions, and Statements

REFERENCE:
HTTPS://WWW.PY4E.COM/HTML3/02-VARIABLES

VARIABLES AND BASICS

CONSTANTS

PYTHON

- 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
```

RESERVED WORDS

PYTHON

- 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
```

VARIABLES

PYTHON

- 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
```

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

NAME YOUR VARIABLES “RIGHT”

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

- What is this bit of code doing??

NAME YOUR VARIABLES "RIGHT"

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

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

- What is this bit of code doing??

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

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

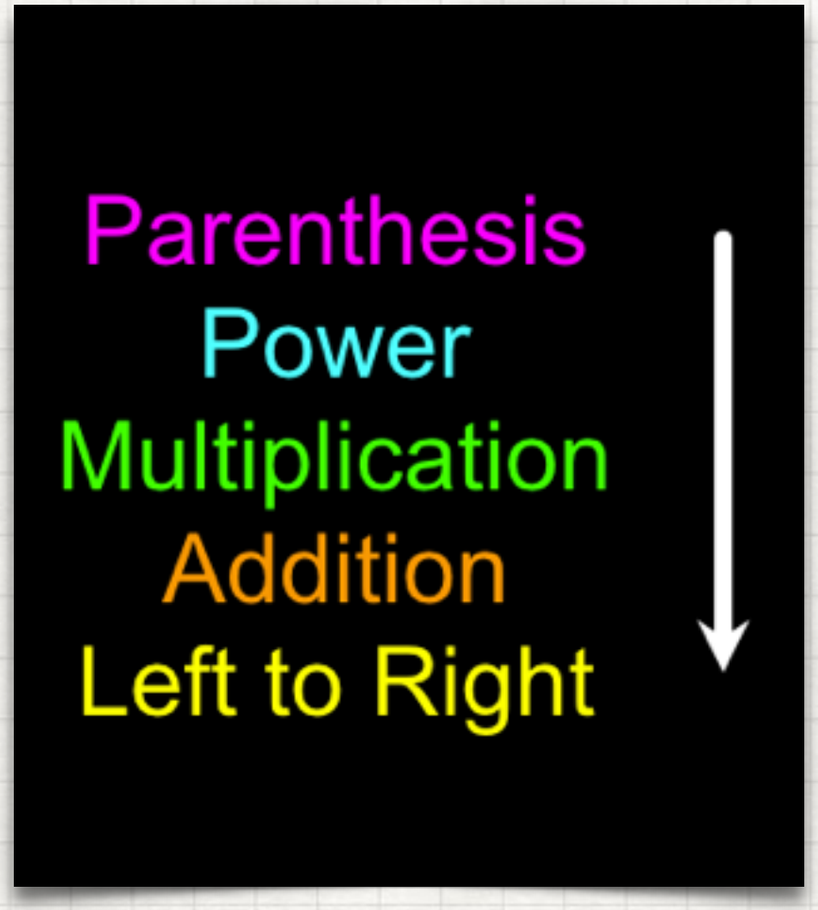
ORDER OF EVALUATION

- 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
```

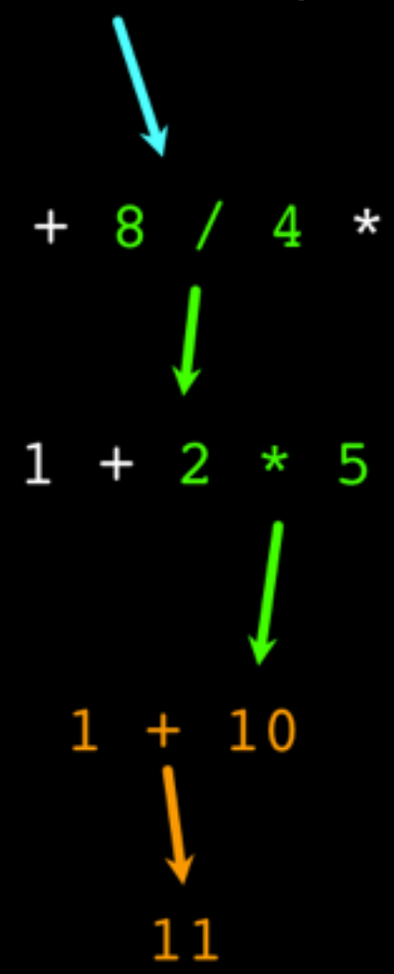
OPERATOR PRECEDENCE RULES

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



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

Parenthesis
Power
Multiplication
Addition
Left to Right



TIPS: OPERATOR

- Remember the rules top to bottom
- If you cannot remember the rulse,
  ALWAYS 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
- We can ask Python what type something is by using the type() function

TYPE MATTERS

- Python knows what “type” everything is
- Some operations are prohibited
- You cannot “add 1” to a string

```
>>> 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'>
>>>
```

INTEGER DIVISION

- Integer division produces a floating point result

- NOTE: this is NOT true for 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

DONEC QUIS NUNC

- 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'
```

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

Even if you may not know how to write python code yet,
But you know what this piece of code is about.

```
# 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)
```

REFERENCE:
HTTPS://WWW.PY4E.COM/HTML3/03-CONDITIONAL

CONDITIONAL EXECUTION

CONDITIONAL STEPS

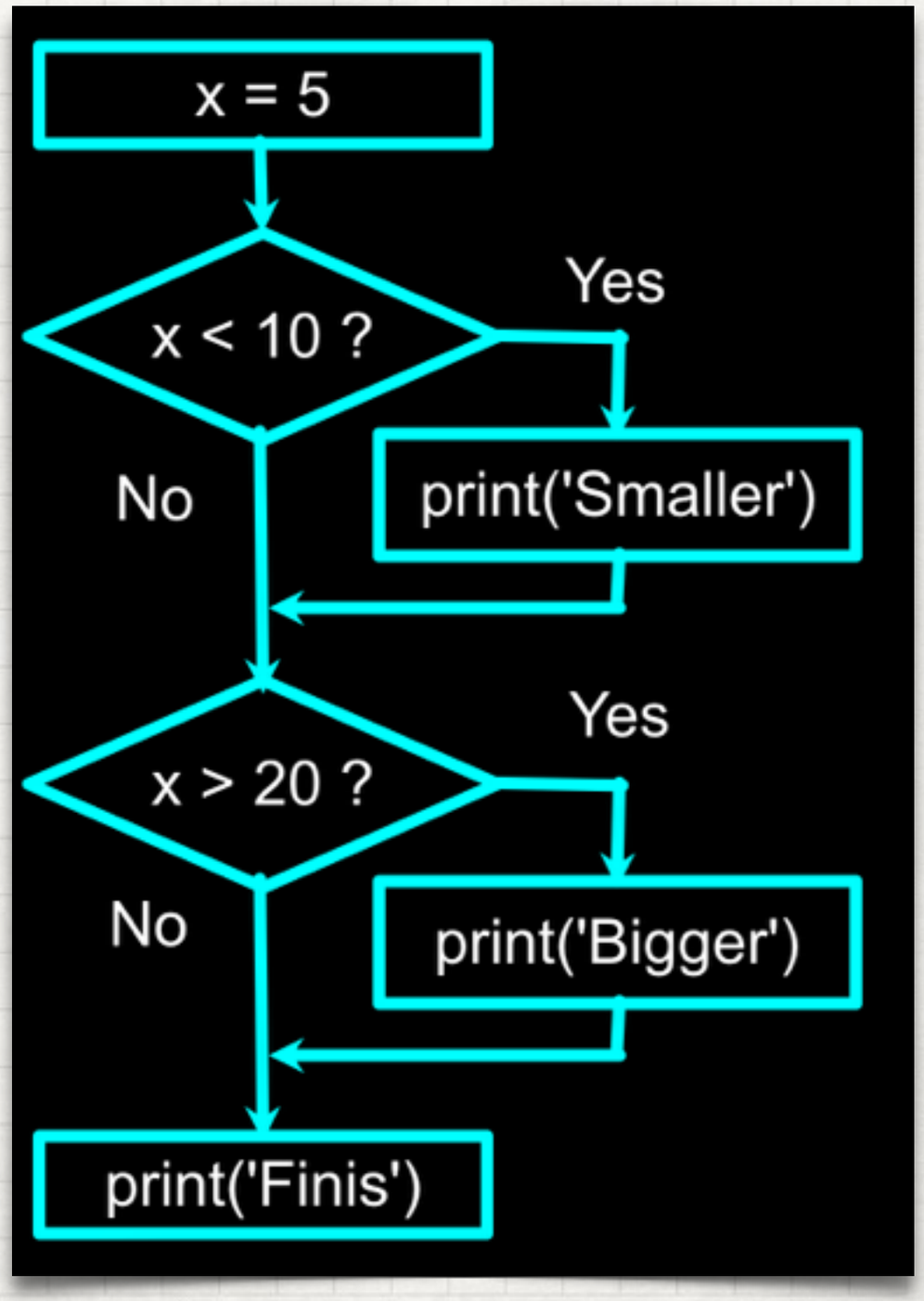
Program:

```
x = 5
if x < 10:
    print('Smaller')
if x > 20:
    print('Bigger')

print('Finis')
```

Output:

```
Smaller
Finis
```



COMPARISON OPERATORS

- Boolean expressions ask a question and produce a Yes or No result which we use to control program flow
- Boolean expressions using comparison operators evaluate to True / False or Yes / No
- Comparison operators look at variables but do not change the variables
- http://en.wikipedia.org/wiki/George_Boole

Python	Meaning
<	Less than
<=	Less than or Equal to
==	Equal to
>=	Greater than or Equal to
>	Greater than
!=	Not equal

INDENTATION

- Increase indent:
  indent after an if statement or for statement (after :)
- Maintain
  indent to indicate the scope of the block (lines are affected by the if/for)
- Reduce indent:
  back to the level of the if statement or for statement to indicate the end of the block
- Blank lines: are ignored - they do not affect indentation
- Comments: on a line by themselves are ignored with regard to indentation

Warning: Turn Off Tabs!!

Python cares A LOT about indentation. If you mix tabs and spaces, you may get "indentation errors" even if everything looks fine

INDENTATION

DONEC QUIS NUNC

only ‘if’ happens

for every i

```
x = 5
if x > 2 :
    print('Bigger than 2')
    print('Still bigger')
print('Done with 2')

for i in range(5) :
    print(i)
    if i > 2 :
        print('Bigger than 2')
    print('Done with i', i)
print('All Done')
```

THINK ABOUT BEGIN/END BLOCKS

```
x = 5
if x > 2 :
    print('Bigger than 2')
    print('Still bigger')
print('Done with 2')

for i in range(5) :
    print(i)
    if i > 2 :
        print('Bigger than 2')
    print('Done with i', i)
print('All Done')
```

NESTED DECISIONS

```
x = 42
if x > 1 :
    print('More than
one')
    if x < 100 :
        print('Less
than 100')
print('All done')
```