Introduction to Python

Part 1: Basic types and control flow

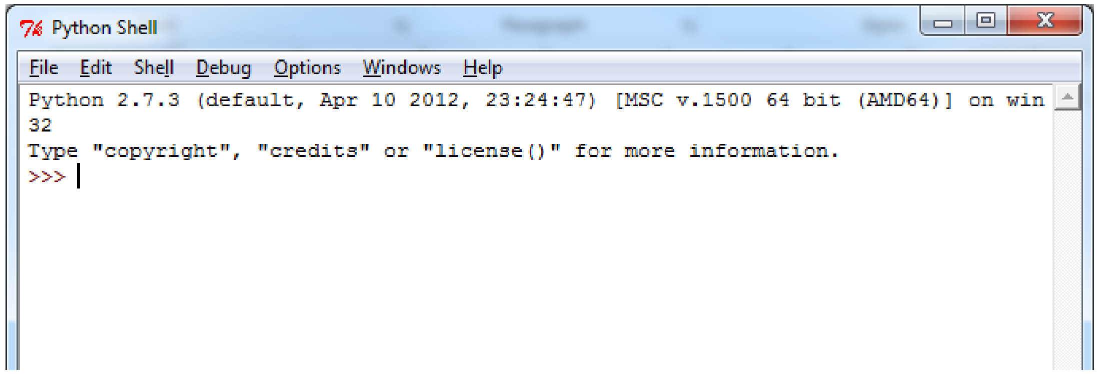
COMP 089H Fall 2015

Intro to Python: part 1

- Intro to IDLE, Python
    - Keyword: print
    - Types: str, int, float
    - Variables
    - User input
    - Saving your work
    - Comments
- Conditionals
    - Type: bool
    - Keywords: and, or, not, if, elif, else

IDLE: **I**nteractive **D**eve**L**opment **E**nvironment

- Shell
  - Evaluates what you enter and displays output
  - Interactive
    - Type at “>>>” prompt
- Editor
  - Create and save .py files
  - Can run files and display output in shell



```
Python 2.7.3 (default, Apr 10 2012, 23:24:47) [MSC v.1500 64 bit (AMD64)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> 
```

Hello, world!

The canonical programming example for a language: write “Hello, world!”

It’s very easy in Python. ☺

```
>>> print “Hello, world!”
Hello, world!
```

Hello, world!

Syntax highlighting:

- IDLE colors code differently depending on functionality
  - Orange: keyword
  - Green: string
  - Blue: output in shell

```
>>> print “Hello, world!”
Hello, world!
```

Hello, world!

Syntax highlighting:

- IDLE colors code differently depending on functionality
  - Orange: keyword
  - Green: string
  - Blue: output in shell

```
>>> print “Hello, world!”
Hello, world!
```

```
# Example keywords:
print if    else  and   or
class while for   break elif
in    def   not   from  import
```

Hello, world!

Syntax highlighting:

- IDLE colors code differently depending on functionality
  - Orange: keyword
  - Green: string
  - Blue: output in shell

```
>>> print “Hello, world!”
Hello, world!
```

```
# Example strings:
“Hello, world!”

‘abc 123 lots of stuff’

“This has ‘nested’ quotes”
```

Hello, world!

Syntax highlighting:

- IDLE colors code differently depending on functionality
  - Orange: keyword
  - Green: string
  - Blue: output in shell

```
>>> print “Hello, world!”
Hello, world!
```

```
# Any time you have output in
# the shell window, IDLE
# colors it blue by default
```

Types: str, int, float

We've already seen one *type* in Python, used for words and phrases.

In general, this type is called "string". In Python, it's referred to as `str`.

```
>>> print "Hello," + " world!"
Hello, world!

>>> print "a" + 'b' + "'c'"
ab'c'
```

Types: str, int, float

Python also has *types* for numbers.

`int` – integers

`float` – floating point (decimal) numbers

```
>>> print 4                  # int
4

>>> print 6.                 # float
6.0

>>> print 2.3914             # float
2.3914
```

Types: str, int, float

When you add two `int`s you get an `int`.

When you add two `float`s or an `int` and a `float`, you get a `float`.

```
>>> print 4 + 6           # int
10

>>> print 4 + 6.          # float
10.0

>>> print 4.0 + 6.0       # float
10.0
```

Types: str, int, float

When you add two `int`s you get an `int`.

When you add two `float`s or an `int` and a `float`, you get a `float`.

This is true for other operations, too.

```
>>> print 6. - 3
3.0
>>> print 2 * 10
20
>>> print 2 / 10.0
0.2
>>> print 7 % 2
1
>>> print 2 ** 3
8
```

Types: str, int, float

When you add two `int`s you get an `int`.

Sometimes this leads to unexpected results when dividing `int`s.

```
>>> print 2 / 3
0
>>> print 3 / 10
3

# Python rounds down because
# the result of integer division
# is also an int.
```

Variables

To re-use a value in multiple computations, store it in a *variable*.

```
>>> a = 2
>>> print a * a
2

>>> b = 5
>>> print a + b
7
```

Variables

To re-use a value in multiple computations, store it in a *variable*.

Python is "dynamically-typed", so you can change the type of value stored.

- unlike Java, C#, C++, …

```
>>> someVar = 2
>>> print someVar  # it's an int
2
>>> someVar = "Why hello there"
>>> print someVar  # now str
Why hello there
```

Variables

There are some restrictions on variable names. They must:

- be at least 1 character long
- contain only A-Z, a-z, 0-9, and _
- not start with a number
- not be a keyword

```
# Okay variable names:
banana
i_am_awesome
studentCount

# Not good:
123aaa
print
```

Variables

There are some restrictions on variable names. They must:

- be at least 1 character long
- contain only A-Z, a-z, 0-9, and _
- not start with a number
- not be a keyword

Also, don’t use __stuff__, this could show up in future versions.

```
# Okay variable names:
banana
i_am_awesome
studentCount

# Not good:
123aaa
print
__bananas__
__student_count__
```

Intro to Python: part 1

- Intro to IDLE, Python
  - Keyword: print
  - Types: str, int, float
  - Variables
  - User input
  - Saving your work
  - Comments
- Conditionals
  - Type: bool
  - Keywords: and, or, not, if, elif, else

User Input

Two choices for *functions*:

- `raw_input`
  - *Returns* a string
  - Very handy – always use this!
- `input`
  - We will not use this. It can be very dangerous if you're not careful.

```
>>> color = raw_input("What is your favorite color? ")
What is your favorite color? Teal

>>> print "Your favorite color is", color
Your favorite color is teal
```

User Input

Functions are procedures you can *call*.

They may or may not *return* a value.

- If they do, you are effectively replacing the *function call* with the result.

```
>>> color = raw_input("What is your favorite color? ")
What is your favorite color? Teal

>>> print "Your favorite color is", color
Your favorite color is teal
```

User Input

Functions are procedures you can *call*.

You *call* a function by putting parentheses after its name.

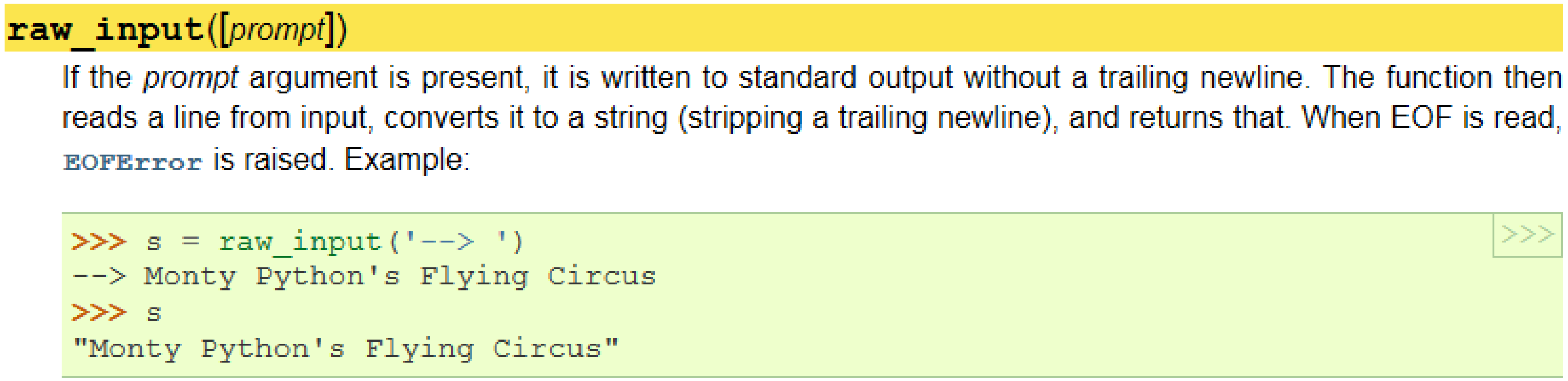
Anything inside the parentheses are *parameters*, separated by commas.

```
>>> color = raw_input("What is your favorite
color? ")
What is your favorite color? Teal

>>> print "Your favorite color is", color
Your favorite color is teal
```

User Input

Functions are procedures you can *call*.

`raw_input` only has one argument, prompt, and it is optional (note the [] in the documentation).



raw_input([*prompt*])

If the *prompt* argument is present, it is written to standard output without a trailing newline. The function then reads a line from input, converts it to a string (stripping a trailing newline), and returns that. When EOF is read, `EOFError` is raised. Example:

```
>>> s = raw_input('--> ')
--> Monty Python's Flying Circus
>>> s
"Monty Python's Flying Circus"
```

See https://docs.python.org/2/library/functions.html

User Input

`print` can take multiple values, separated by commas

- It replaces each comma with a space

```
>>> color = raw_input(“What is your favorite color? ”)
What is your favorite color? Teal

>>> print “Your favorite color is”, color
Your favorite color is teal

>>> print 4, 2, 9
4 2 9
```

User Input

`print` can take multiple values, separated by commas

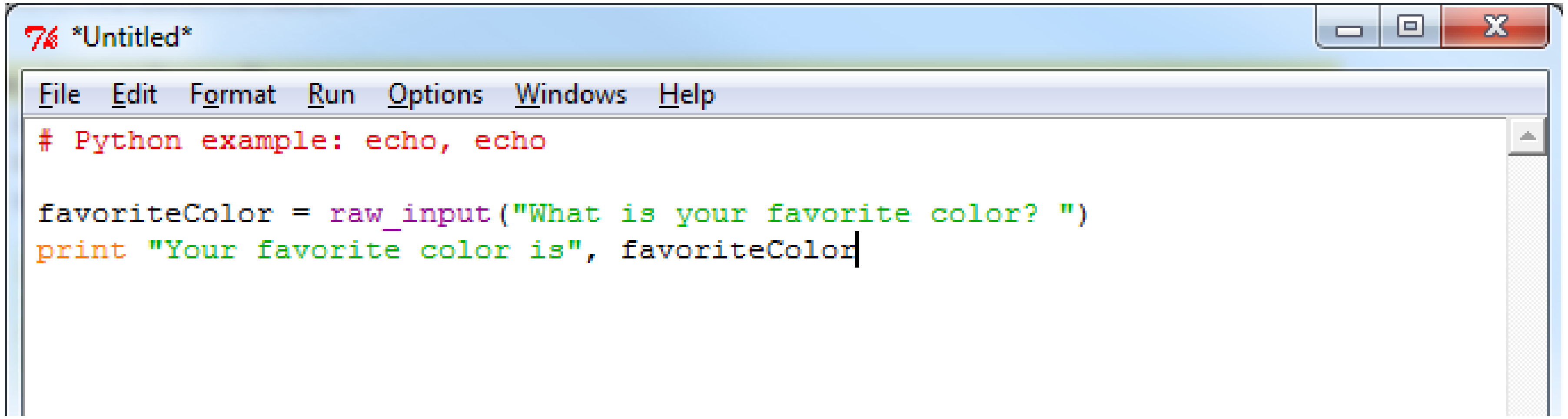
- It replaces each comma with a space

If you don’t want spaces, use the built-in `str` function to convert values to strings, then add them.

```
>>> color = raw_input(“What is your favorite
color? ”)
What is your favorite color? Teal

>>> print “Your favorite color is”, color
Your favorite color is teal

>>> print 4, 2, 9
4 2 9

>>> print “a” + str(1) + “b” + str(2)
a1b2
```

Creating a .py file

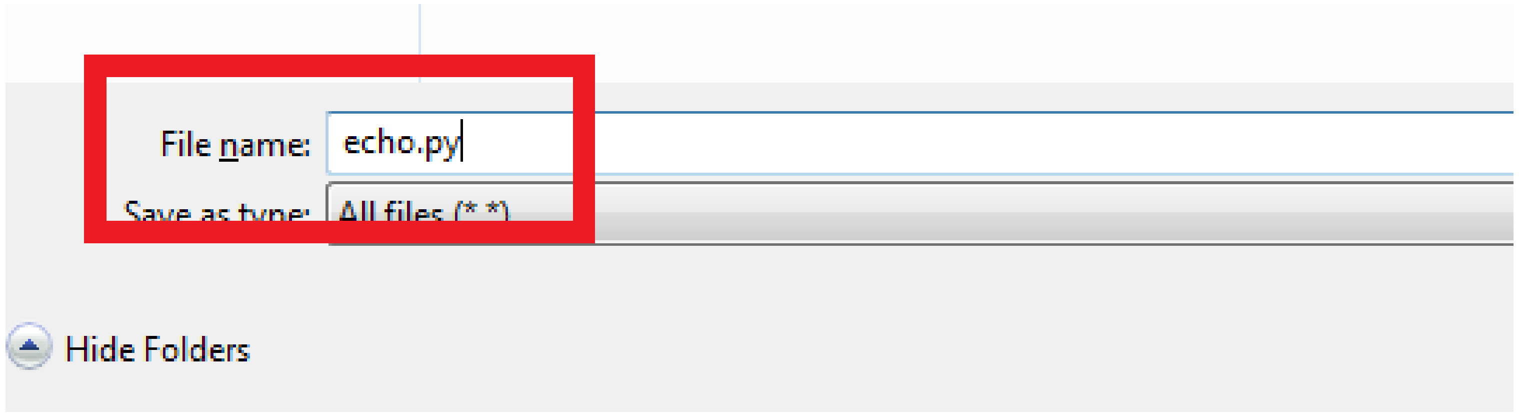
- File -> New Window



```
# Python example: echo, echo

favoriteColor = raw_input("What is your favorite color? ")
print "Your favorite color is", favoriteColor
```

Creating a .py file

- File -> New Window
- Make sure you enter .py as the file extension; IDLE doesn't always do this, and you will lose syntax highlighting ☹

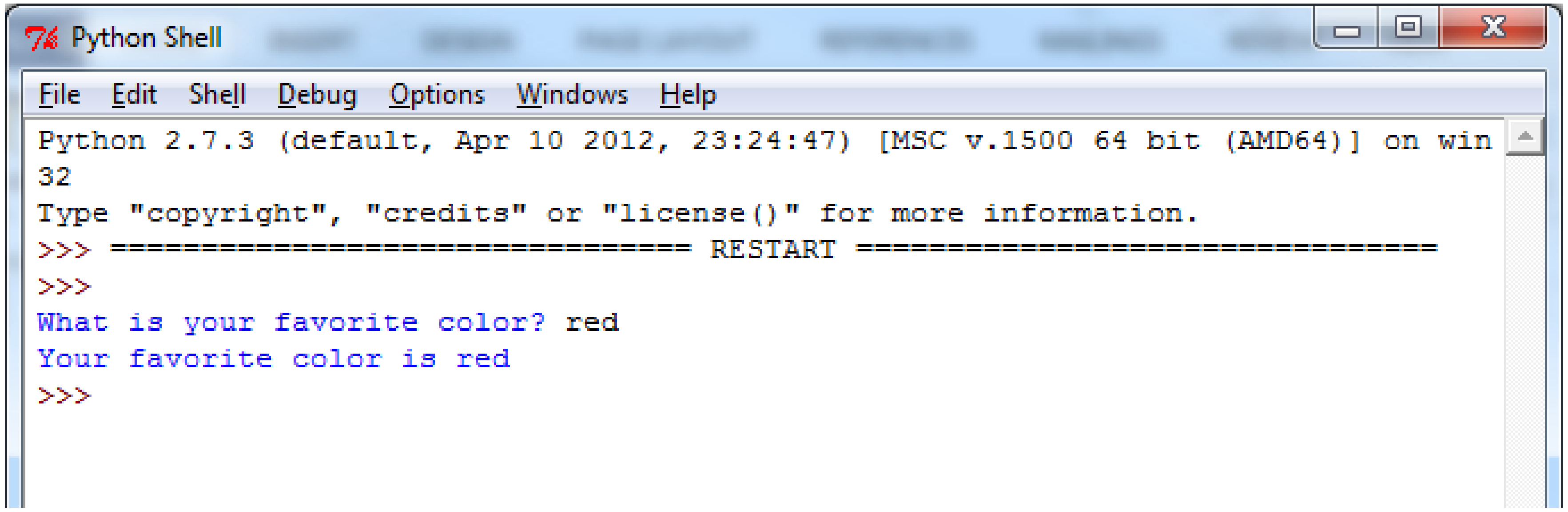


File name: echo.py

Save as type: All files (*.*)

Creating a .py file

- File -> New Window
- Make sure you enter .py as the file extension; IDLE doesn't always do this, and you will lose syntax highlighting ☹
- Go to Run -> Run Module (F5) to save and run your program



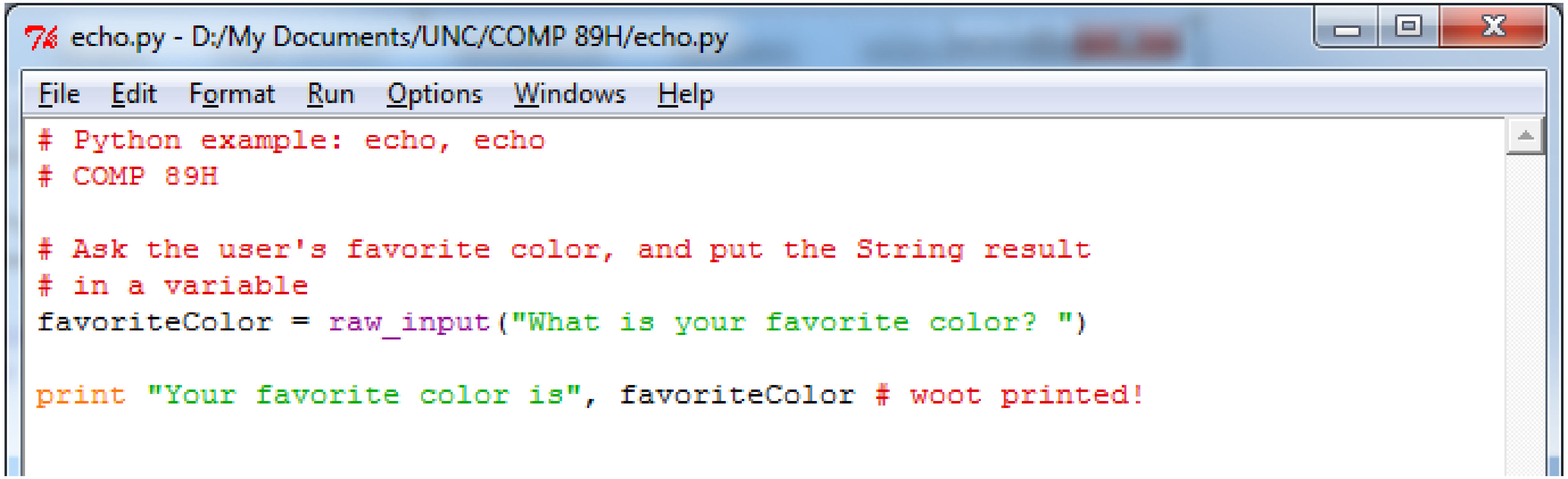
```
Python 2.7.3 (default, Apr 10 2012, 23:24:47) [MSC v.1500 64 bit (AMD64)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
What is your favorite color? red
Your favorite color is red
>>>
```

## Comments

# You’ve already seen some!

# Comments in Python are denoted with a #, and are colored red

# They aren’t run, and are used to help with readability


echo.py - D:/My Documents/UNC/COMP 89H/echo.py
File Edit Format Run Options Windows Help

```
# Python example: echo, echo
# COMP 89H

# Ask the user's favorite color, and put the String result
# in a variable
favoriteColor = raw_input("What is your favorite color? ")

print "Your favorite color is", favoriteColor # woot printed!
```

Intro to Python: part 1

- Intro to IDLE, Python
  - Keyword: print
  - Types: str, int, float
  - Variables
  - User input
  - Saving your work
  - Comments
- Conditionals
  - Type: bool
  - Keywords: and, or, not, if, elif, else

Type: bool

Boolean values are true or false.

Python has the values `True` and `False` (note the capital letters!).

You can compare values with ==, !=, <, <=, >, >=, and the result of these expressions is a `bool`.

```
>>> a = 2
>>> b = 5
>>> a > b
False
>>> a <= b
True
>>> a == b # does a equal b?
False
>>> a != b # does a not-equal b?
True
```

Type: bool

When combining Boolean expressions, parentheses are your friends.

```
>>> a = 2
>>> b = 5
>>> False == (a > b)
True
```

Keywords: and, or, not

`and` is `True` if both parts evaluate to `True`, otherwise `False`

`or` is `True` if at least one part evaluates to `True` , otherwise `False`

```
>>> a = 2
>>> b = 5
>>> a < b and False
False
>>> a < b or a == b
True
>>> a < b and a == b
False
>>> True and False
False
>>> True and True
True
>>> True or False
True
```

Keywords: and, or, not

`and` is `True` if both parts evaluate to `True`, otherwise `False`

`or` is `True` if at least one part evaluates to `True` , otherwise `False`

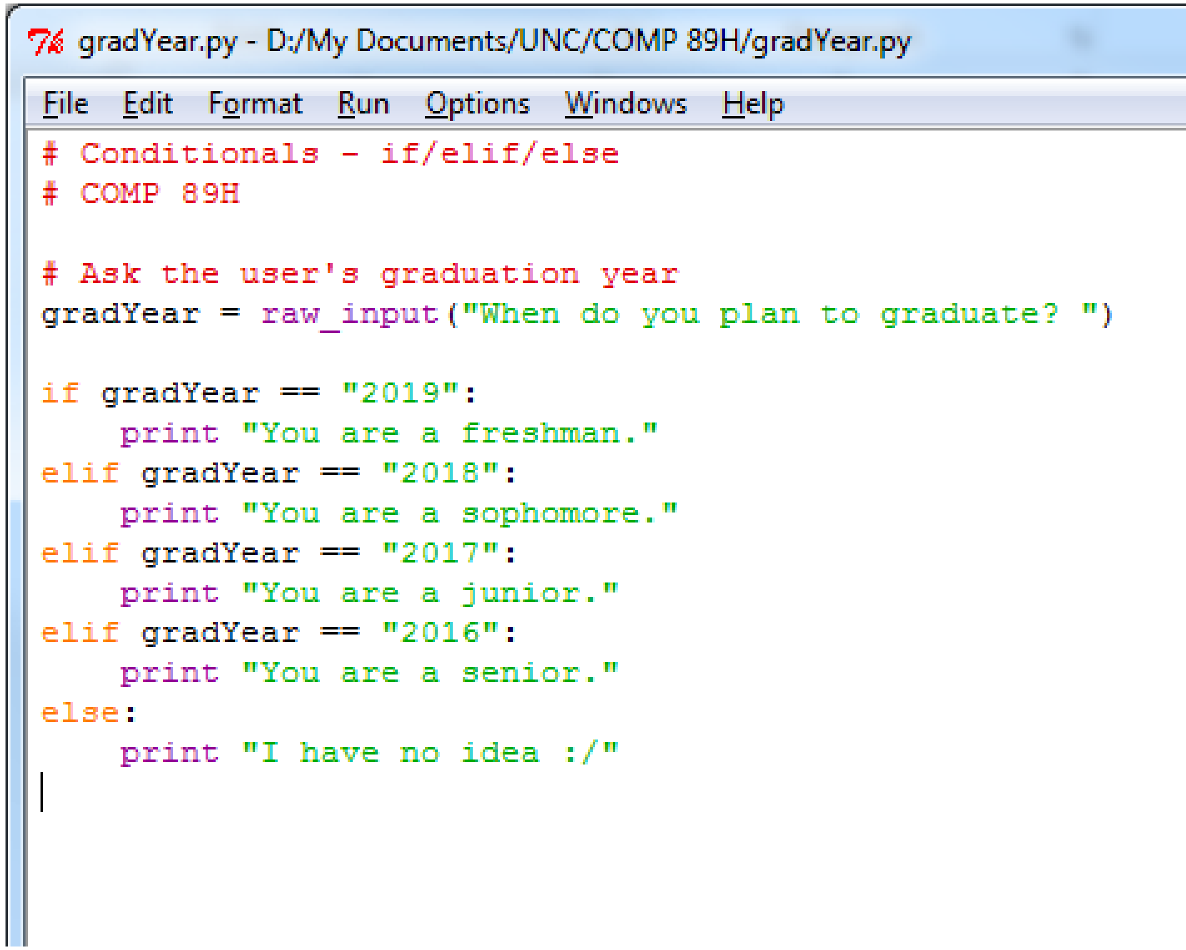
`not` is the opposite of its argument

```
>>> not True
False
>>> not False
True
```

```
>>> True and (False or not True)
False
>>> True and (False or not False)
True
```

Conditionals: if, elif, else

The keywords `if`, `elif`, and `else` provide a way to control the flow of your program.

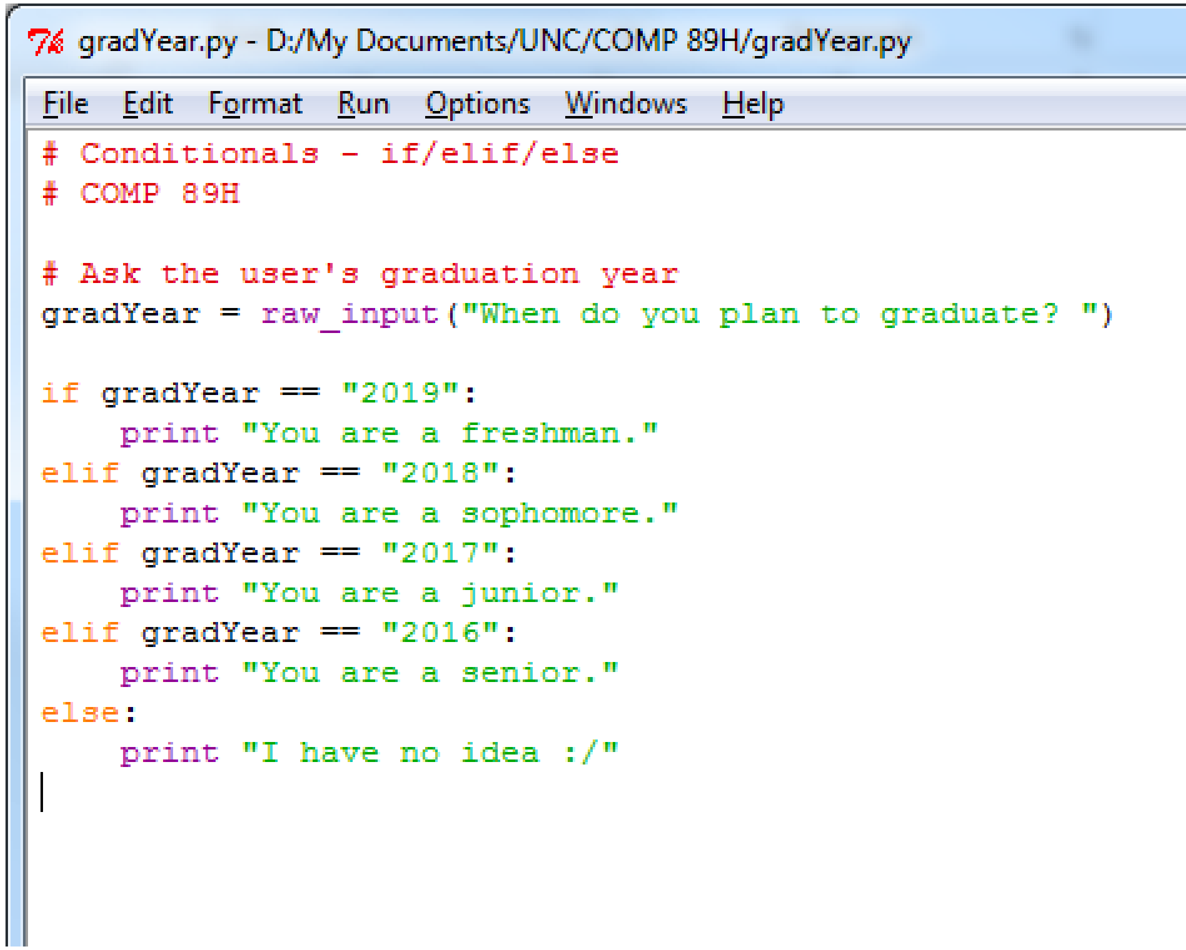

gradYear.py - D:/My Documents/UNC/COMP 89H/gradYear.py
File Edit Format Run Options Windows Help

```
# Conditionals - if/elif/else
# COMP 89H

# Ask the user's graduation year
gradYear = raw_input("When do you plan to graduate? ")

if gradYear == "2019":
    print "You are a freshman."
elif gradYear == "2018":
    print "You are a sophomore."
elif gradYear == "2017":
    print "You are a junior."
elif gradYear == "2016":
    print "You are a senior."
else:
    print "I have no idea :/"
```

Conditionals: if, elif, else

The keywords `if`, `elif`, and `else` provide a way to control the flow of your program.

Python checks each condition in order, and executes the block (whatever's indented) of the first one to be `True`.

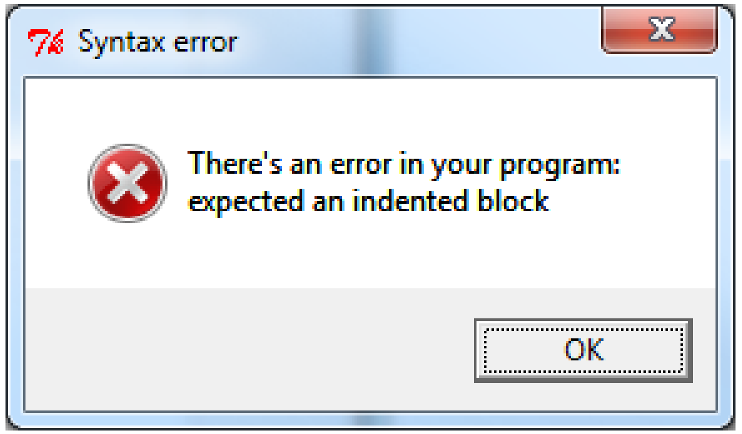

gradYear.py - D:/My Documents/UNC/COMP 89H/gradYear.py
File Edit Format Run Options Windows Help

```
# Conditionals - if/elif/else
# COMP 89H

# Ask the user's graduation year
gradYear = raw_input("When do you plan to graduate? ")

if gradYear == "2019":
    print "You are a freshman."
elif gradYear == "2018":
    print "You are a sophomore."
elif gradYear == "2017":
    print "You are a junior."
elif gradYear == "2016":
    print "You are a senior."
else:
    print "I have no idea :/"
```

Conditionals: if, elif, else

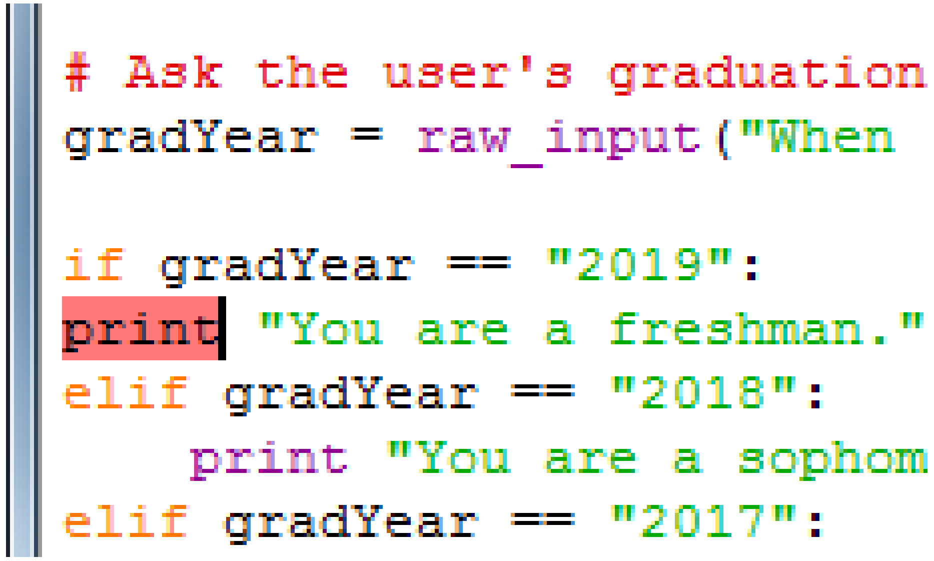
Indentation is important in Python!

Make sure each `if`, `elif`, and `else` has a colon after it, and its block is indented one tab (4 spaces by default).



Syntax error

There's an error in your program:
expected an indented block



```
# Ask the user's graduation
gradYear = raw_input("When

if gradYear == "2019":
print "You are a freshman."
elif gradYear == "2018":
    print "You are a sophom
elif gradYear == "2017":
```

Conditionals: if, elif, else

Make sure you're careful what you compare to the result of `raw_input`. It is a string, not a number.

```
# The right way: str to str or int to int
>>> gradYear = raw_input("When do you plan to graduate? ")
When do you plan to graduate? 2019
>>> gradYear == 2019 # gradYear is a string :(
False
>>> gradYear == "2019"
True
>>> int(gradYear) == 2019 # cast gradYear to an int :)
True
```

Conditionals: if, elif, else

Make sure you're careful how to compare the result of `raw_input`. It is a string, not a number.

Doing it wrong leads to a `ValueError`:

```
>>> gradYear = raw_input("When do you plan to graduate? ")
When do you plan to graduate? Sometime
>>> int(gradYear) == 2019

Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    int(gradYear) == 2019
ValueError: invalid literal for int() with base 10: 'sometime'
```

Today we covered:

- Intro to IDLE, Python
    - Keyword: print
    - Types: str, int, float
    - Variables
    - User input
    - Saving your work
    - Comments
- Conditionals
    - Type: bool
    - Keywords: and, or, not, if, elif, else