We are 183

L19: Week 12 – Monday

Reminders!

- Assignment 5 due Friday, April 1
- Final Project Core due a week from Friday, April 8

Today in EECS183:
Introduction to

Why Python?

- It's really good to know more than one language

- The thought processes and logic you learned in C++ will also work in Python

- In fact, they'll work for almost any programming language!

Python and C++ are used for different things

- Python is **interpreted** rather than **compiled**
- It is usually faster to develop in Python
- You can accomplish more with fewer lines of code
- Many scripts **for this class** were written in Python (e.g., autograder)
- **But**, Python is less structured and less efficient

Python and C++ are used for different things

- Python is **interpreted** rather than **compiled**
- It i
- Yo code
- Ma ython (e.
- **But**, Python is less structured and less efficient

Python is great for getting a program up-and-running **quickly**

development speed > program speed

Python and C++ are used for different things

- Python is **interpreted** rather than **compiled**
- It i
- Yo code
- Ma ython (e.
- **But**, Python is less structured and less efficient

C++ is great for building **large programs** and **maximizing performance**

development speed < program speed

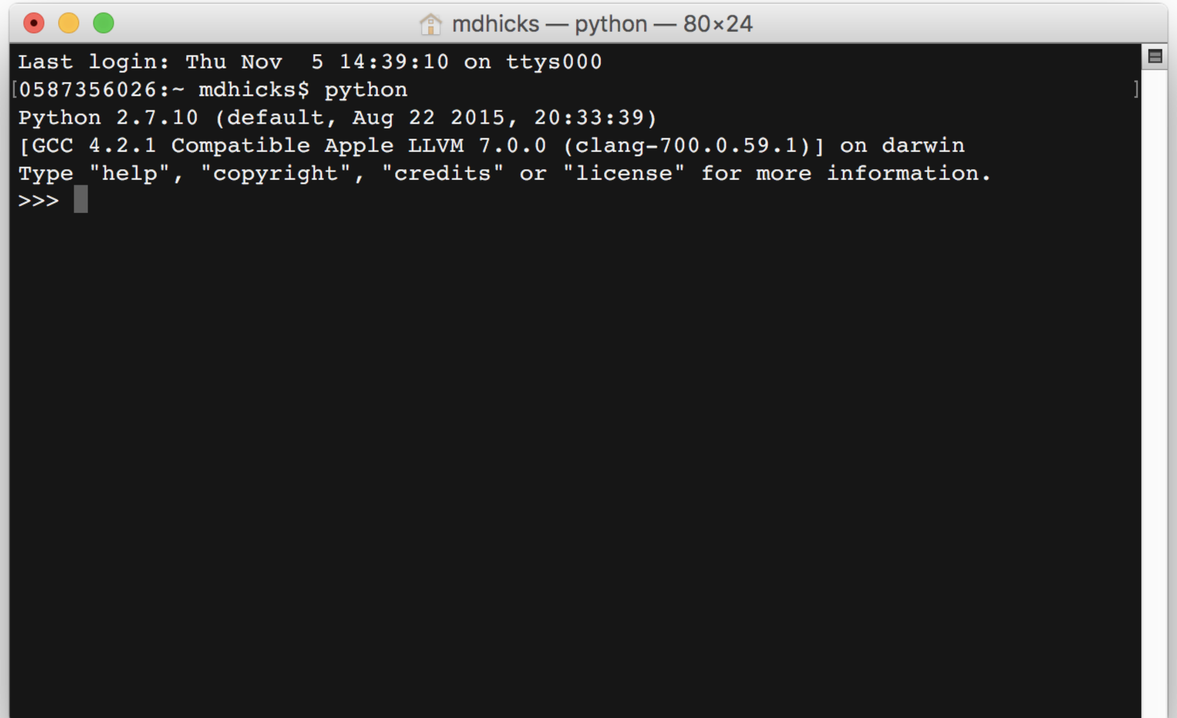
Python Version

- Make sure that you are using Python 2.x, not 3.x
  - Version 3.x uses different syntax, and is not backwards compatible

- Many computers on campus have 2.7 installed

Getting Started

- Mac Users
  - Open Terminal Window
  - Type python



```
Last login: Thu Nov  5 14:39:10 on ttys000
[0587356026:~ mdhicks$ python
Python 2.7.10 (default, Aug 22 2015, 20:33:39)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Getting Started

- PC Users
  - – Install “python”
  - – Python.org
  - – Open up the “Downloads” menu at the top
    - Choose “Download Python 2.7.10”
    - Do NOT choose to install any version starting with 3

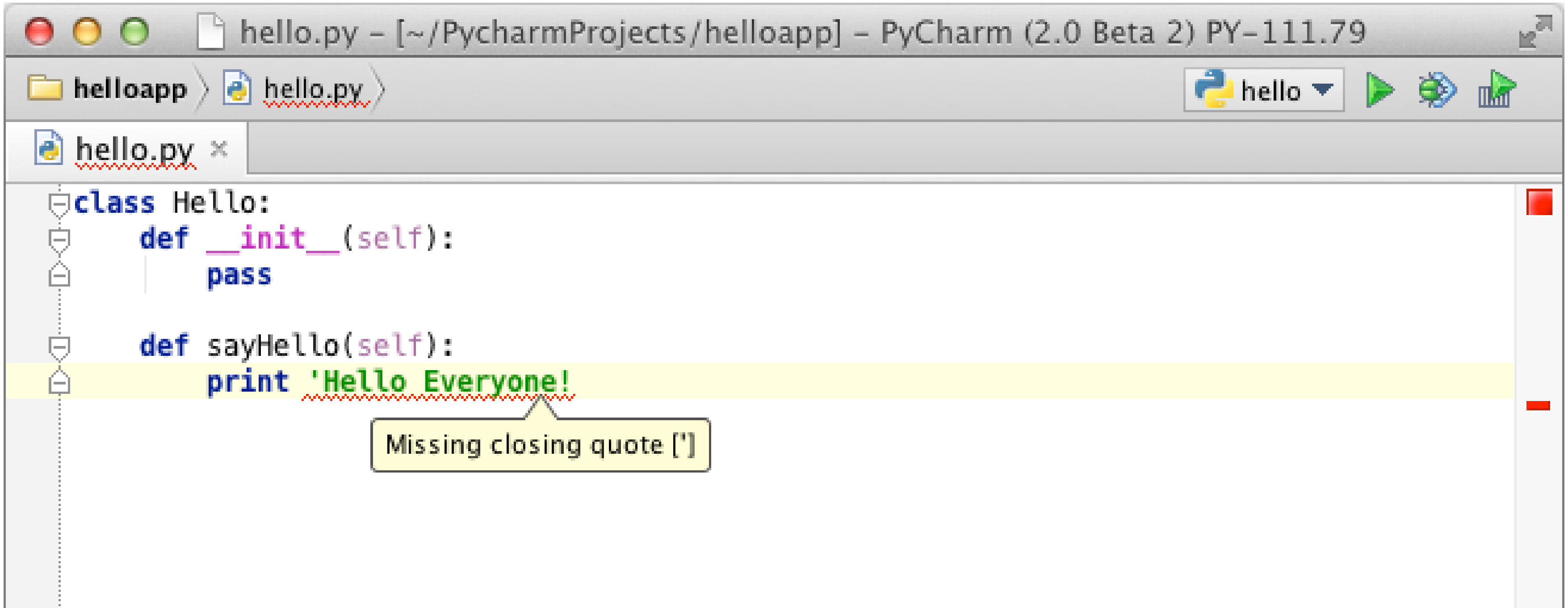
Really good PC Interface, more requirements

- https://pytools.codeplex.com/wikipage?title=PTVS%20Installation

- Requires:
  - Visual Studio
    - Pro (NOT Express), or free shell from the link above
  - Python already installed
- Allows:
  - Use of the Visual Studio interactive debugger

Getting Started – PyCharm

- PyCharm is a free, cross-platform Python IDE
  - – https://www.jetbrains.com/pycharm/

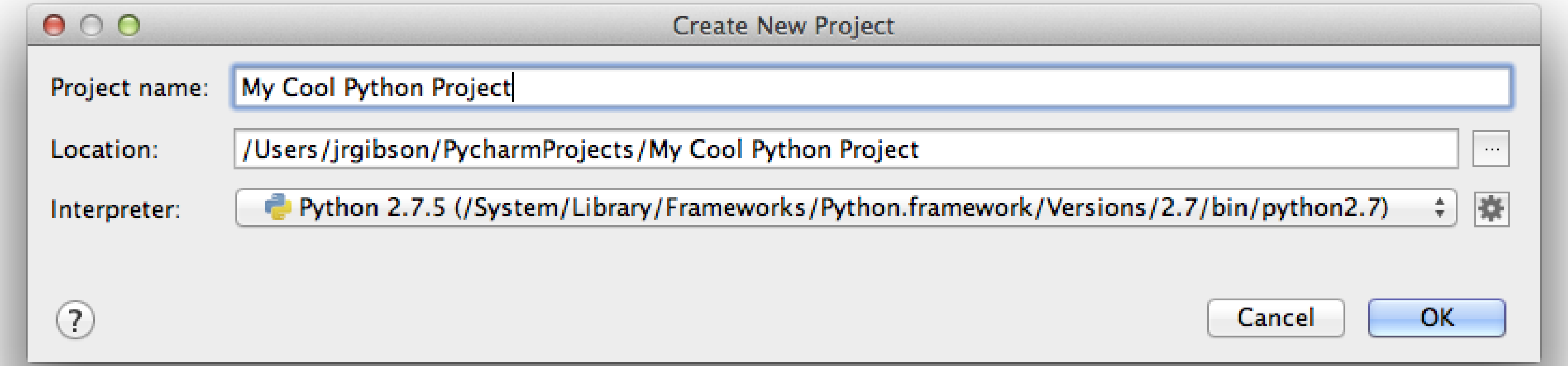

hello.py - [~/PycharmProjects/helloapp] - PyCharm (2.0 Beta 2) PY-111.79
helloapp
hello.py
hello
hello.py

```
class Hello:
    def __init__(self):
        pass

    def sayHello(self):
        print 'Hello Everyone!
```

Missing closing quote [']

Getting Started – PyCharm

- When creating a project
  - – Be sure to set the Interpreter to Python 2.7



Create New Project

Project name: My Cool Python Project

Location: /Users/jrgibson/PycharmProjects/My Cool Python Project

Interpreter: Python 2.7.5 (/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7)

>>>   prompt

- Interactive Interpreter prompt
- Don’t type it

- **>>>** is there “prompting” you that this is where you input code

- To get out
  - <ctrl> + D

Useful Python Online Resources

- An online Python interpreter
  - http://repl.it/languages/Python

- Google has a free online Python class
  - https://developers.google.com/edu/python/

First Program

```
>>> print 'Hello World!'
```

Console

```
Hello World!
```

Single vs. Double Quotes

- Python accepts either
  - `name = 'Ann'`
  - `ch = "X"`

- C++ programmers tend to use " to enclose strings and ' for single characters

- Python programmers tend to use ' for everything

# Comments

```
>>> # My first Program
>>> # Author:   My Name
>>> # Date:   11-09-2015
>>> print 'Hello World!'

Hello World!

>>> 1 + 4

5

>>> # I love this – super simple
>>>
```

Multi line comments use ' ' '

```
'''
This is a multi-line comment
that continues onto a second line.
And even onto a third line.
NOTE: the first and last line are triple-quotes
You can use single or double quotes
'''
```

These are generally used to document functions, not within functions

i>Clicker #1

Which of the following are valid ways to start comments in Python?

A) #
B) '''
C) """
D) All of the above
E) None of the above

i>Clicker #1

Which of the following are valid ways to start comments in Python?

A) #

B) '''

C) """

D) All of the above

E) None of the above

Literals

- Hardcoded values
  - – Also known as “literals”

```
>>> 1 + 2  # 1 and 2 are integer literals
3
>>> print 'Hello World!' # string literal

Hello World!
```

Arithmetic Operators

- Common operators are mostly the same as C++:

+ - * / %

- Except:

** Exponentiation

```
>>> print ( 5 ** 2 )
```

```
25
```

Operator Precedence

Precedence	Operator	Grouping
1	( )	Left to right
2	** (exponentiation)	Right to left
3	+ - (unary), cast Example: +2, -3	Right to left
4	* / %	Left to right
5	+ - (binary) Example: 3-2	Left to right
6	=	Right to left

- Grouping defines the precedence order when several operators of the same precedence level are in an expression.

Division is similar to C++

Watch out if you have `int / int` (will floor)

```
2.0 / 3.0  ->  0.6666…

2 / 3  ->  0

5.0 / 2  ->  2.5

5 / 2  ->  2
```

Same behavior as C++

Division is similar to C++

Watch out if you have `int / int` (will floor)

```
2.0 / 3.0  ->  0.6666…

2 /

5.0
```

-5 / 2 -> -3

Python “floors” towards next negative

(C++ truncates)

```
5 / 2  ->  2
```

i>Clicker #2

```
>>> 5 + 2 * 2 ** 2
```

What is the result of executing the above code?

A) 196
B) 81
C) 21
D) 13

i>Clicker #2

```
>>> 5 + 2 * 2 ** 2
```

What is the result of executing the above code?

A) 196
B) 81
C) 21
[✓ D) 13]

Operations on Strings

- **Cannot** perform MOST mathematical operations on strings
  - `'2' - '1'`
  - `'eggs' / 'over easy'`
  - `'third' * 'a charm'`

All of these are illegal!

Operations on Strings

- **Cannot** perform MOST mathematical operations on strings
  - `'2' - '1'`
  - `'eggs' / 'over easy'`
  - `'third' * 'a charm'`

String not char

All of these are illegal!

Illegal in C++ also

Operations on Strings: Concatenation

```python
first = 'Muddy'
second = ' the Mudhen mascot'
print first + second
```

Console

```
Muddy the Mudhen mascot
```

Operations on Strings: Repetition

```
first = 'Muddy'
print first * 3
```

Console

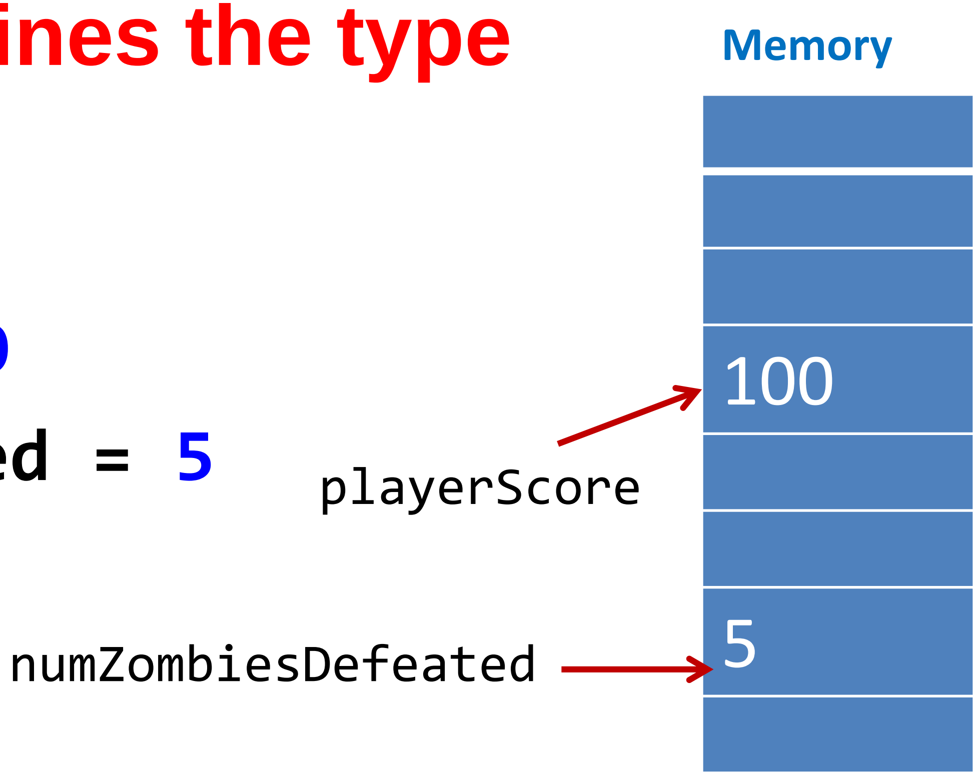
```
MuddyMuddyMuddy
```

Variable Declaration

- Give it a name that describes its purpose
- Give it a value
- **The value determines the type**

- Examples:

```
playerScore = 100
numZombiesDefeated = 5
```



Variables

- Variables must have a name
  - (rules on next slide)

- Python does not allow you to declare variables ahead of time
  - **Assigning a value creates the variable**
  - Python determines the type based on the value to the right of the =

Variable / Identifier Rules

1) Start with a letter (or underscore)

2) After the first character, any number of letters, underscores, or digits

3) Can’t be a keyword/reserved word

- Python variable names are unlimited in length
- Case is significant
- Special identifiers start (and may end) with '_'

Keywords in Python

and
as
assert
break
class
continue
def
del
elif
else
except
exec
finally
for
from
global
if
import
in
is
lambda
not
or
pass
print
raise
return
try
while
with
yield

Keywords we’ll cover

and	del	from	**not**	**while**
as	**elif**	global	**or**	**with**
assert	**else**	**if**	pass	yield
break	except	**import**	**print**	
class	exec	**in**	raise	
continue	finally	is	**return**	
def	**for**	lambda	try	

Data Types

- integer (`int`)
  - long
- float
- string (`str`)
- boolean

Value determines data type

- The assignment determines the data type

```
age = 19  # age refers to an int

age = 5.3 # age now refers to a float
```

Data Types - int

- Examples: `5`, `-1`, `323`, `1000`
- Range:
  - `-2147483468 to 2147483467` (32-bit)
  - Roughly ±2 Billion
- Size: 4 bytes to store the `int` (but Python has more overhead to track the variable)

What happens if you go over that 2 billion?

- In Python, not much

- Python changes data types and stores it as a

  `long`

Console

```
>>> x = 2 ** 31
>>> x = x * 2
>>> x
4294967296
```

Data Types - int

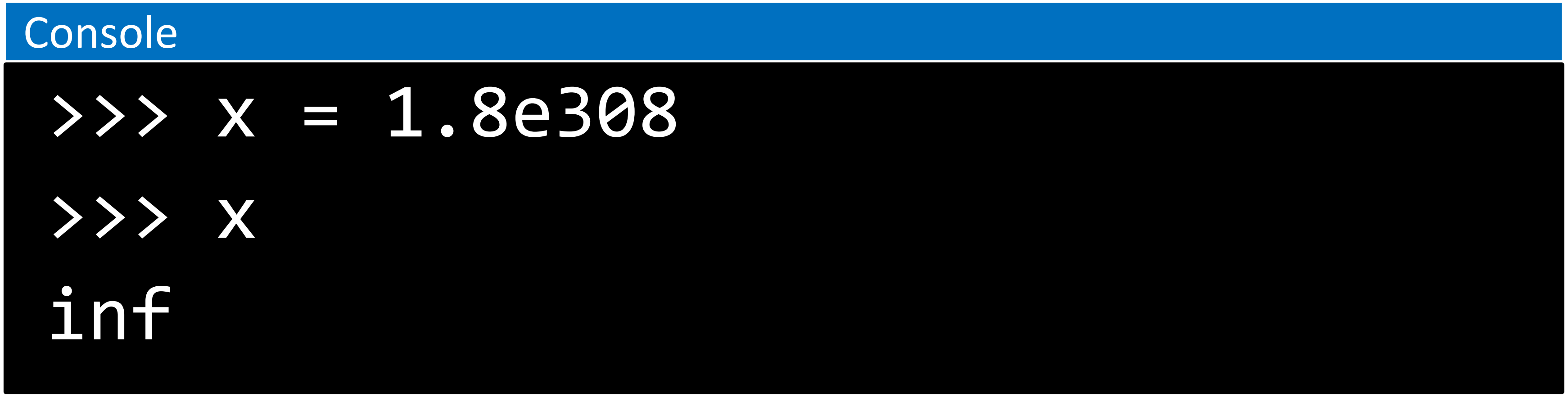
- Whole numbers
- Don't start with a zero (`0123` gives strange results)
  - It's base 8, so `010` is `8` & `011` is `9`
  - `0x` is Hexadecimal, so `0x10` is `16` & `0xff` is `255`
- No commas (1,000 gives odd results!)
- No spaces (1 000 000 won’t work!)

Data Types - float

- Range: `±2.22507e-308 to ±1.79769e+308`
  - Implementation-dependent, some versions of Python might be different

- "default" data type for real numbers
  - Usually equivalent to C++ `double`

What happens if you go over that 1.79769e+308?



```
>>> x = 1.8e308
>>> x
inf
```

This is called overflow

Data Types - boolean

- booleans only have two values

```
True
False
```

- boolean values normally are the result of comparing two values

- More on comparisons coming up soon

i>Clicker #3

```
>>> x = 1000
>>> y = 2000
>>> z = x * y * x * 2
```

Will there be an overflow?

A) yes
B) no

i>Clicker #3

```
>>> x = 1000
>>> y = 2000
>>> z = x * y * x * 2
```

Will there be an overflow?

A) yes

[✓ B) no]

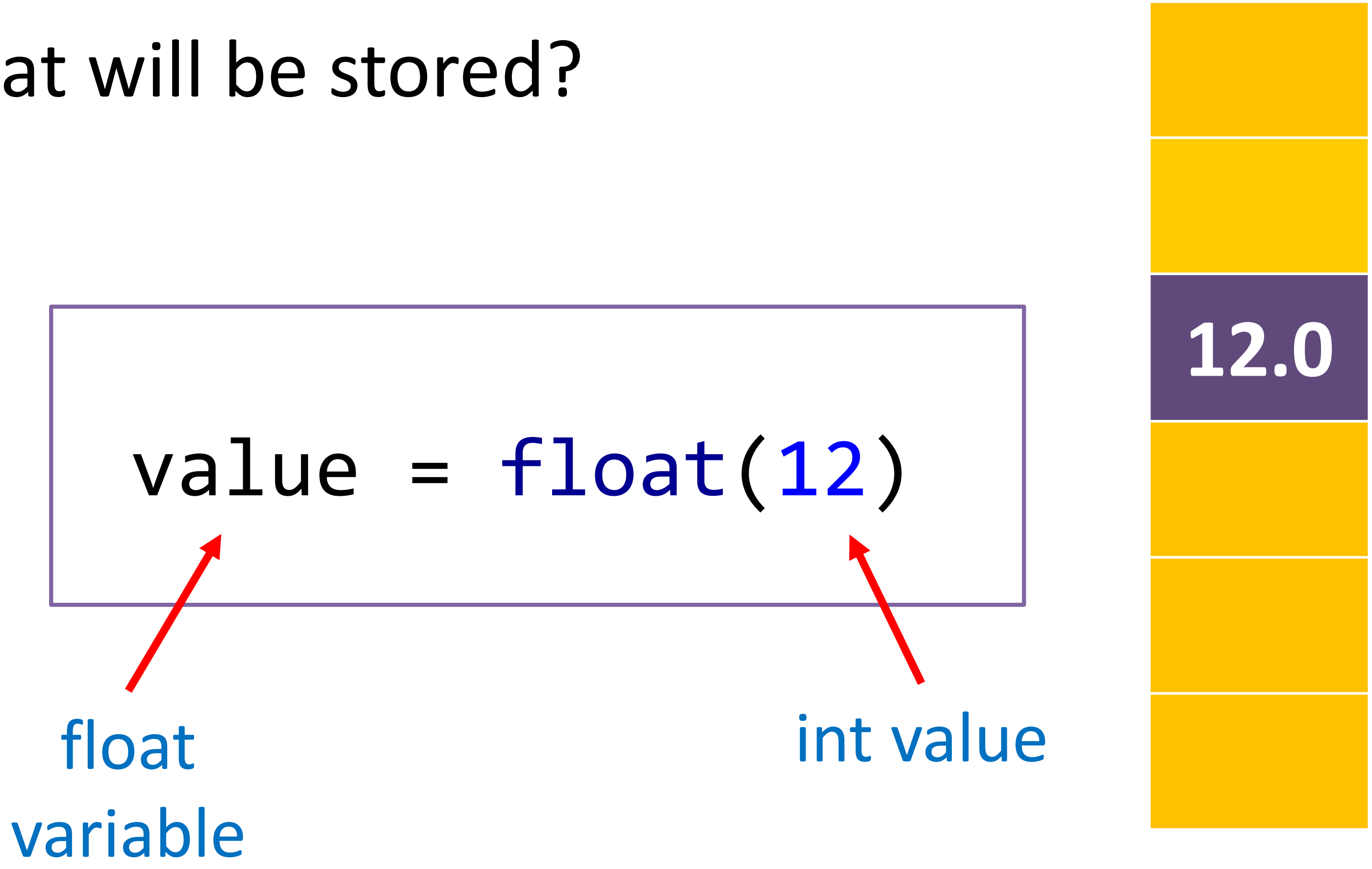
Mixed Mode (Implicit casting)

- Mixed Data Types in expression:
  - Each sub-expression is *promoted* to the *highest* type prior to evaluation
    - In the expression `2 * 3.5`, the `2` is promoted to a `float`

- Type Promotion Guidelines
  - **`int`** is *promoted* to **`long`** is *promoted* to **`float`**

Type Conversion (Explicit casting)

What will be stored?



```
value = float(12)
```

float variable

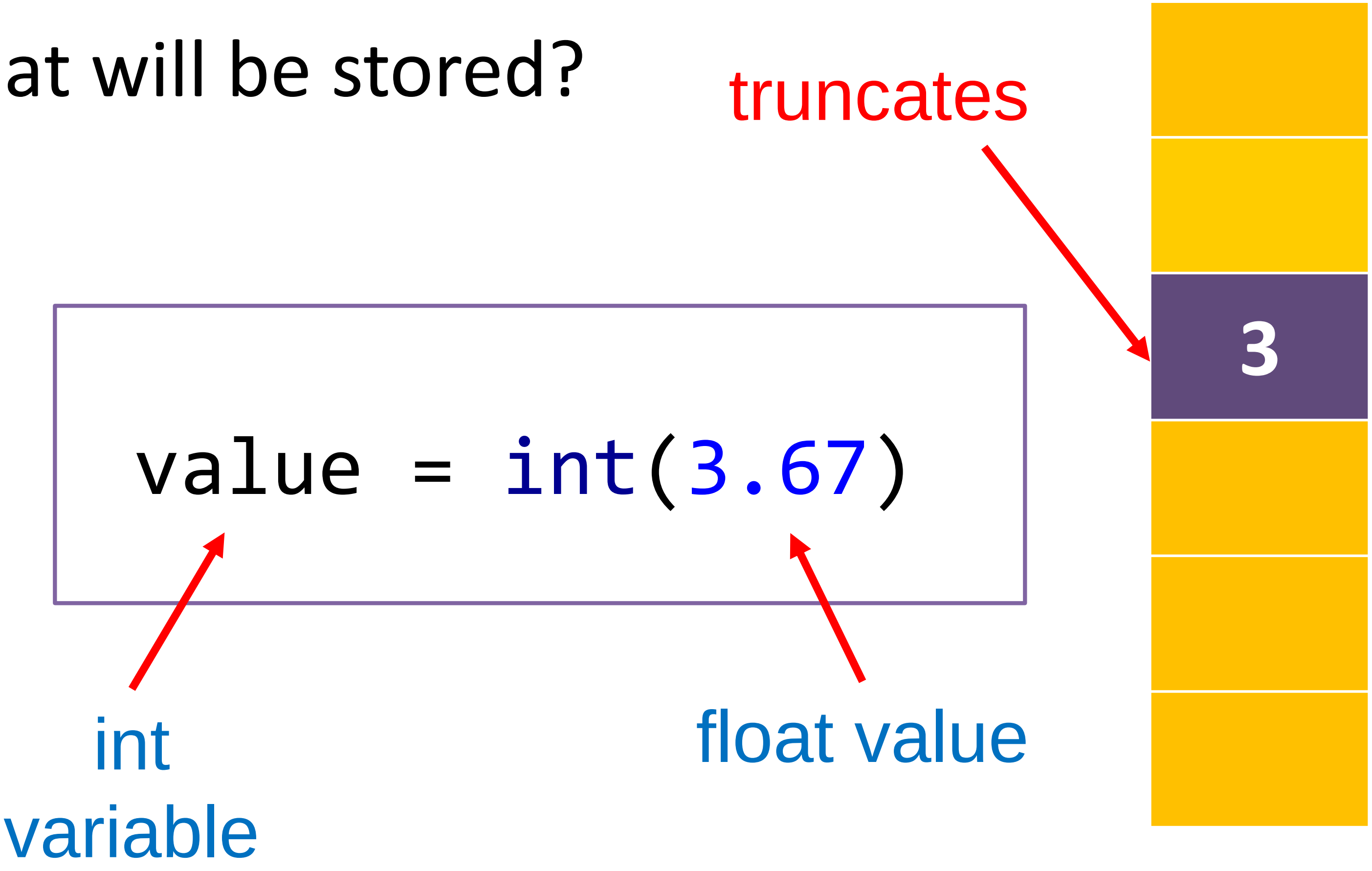
int value

12.0

Explicit type conversion from int to float (**upcasting**)

Type Conversion (Explicit casting)

What will be stored?



```
value = int(3.67)
```

truncates

int variable

float value

Explicit type conversion from float to int (**downcasting**)

Division by zero error

```
i = 5
j = 0
x = i / j
```

```
ZeroDivisionError:
integer division or
modulo by zero
```

Casting and rounding

```
x = 3.7
# truncates: 3
i = int(x)
```

Casting and rounding

```
x = 3.7
# truncates: 3
i = int(x)
# one method to round x: 4
i = int(x + 0.5)
```

Casting and rounding

```
x = 3.7
# truncates: 3
i = int(x)
# one method to round x: 4
i = int(x + 0.5)
# another way to round x: 4.0
i = round(x)
```

Casting and rounding

```
x = 3.7
# truncates: 3
i = int(x)
# one method to round x: 4
i = int(x + 0.5)
# another way to round x: 4.0
i = round(x)
# x, rounded, and cast to int: 4
i = int(round(x))
```

Other Conversions – ord, chr

```
print ord('A')
```

Console

```
65
```

Gives the ASCII value

Other Conversions – ord, chr

```
print ord('A')
```

```
Console
65
```

Gives the ASCII value

```
print ord('A') + 2
```

```
Console
67
```

Other Conversions – ord, chr

```
print ord('A')
```

```
Console
65
```

Gives the ASCII value

```
print ord('A') + 2
```

```
Console
67
```

```
print chr(ord('A') + 2)
```

```
Console
C
```

Gives the ASCII character

i>Clicker #4

```
>>> x = 3.14
>>> x = int(x)
```

What is the type and value of x?

A) float, 3.14
B) float, 3.0
C) int, 3
D) int, 4

i>Clicker #4

```
>>> x = 3.14
>>> x = int(x)
```

What is the type and value of x?

A) float, 3.14
B) float, 3.0
C) int, 3
D) int, 4

Standard I/O Streams

- Standard Output Stream: `print`

```
print 'Hello'
```

- Standard Input Stream: `raw_input()`

```
print 'Enter the first number:',
age = raw_input()
```

print Examples

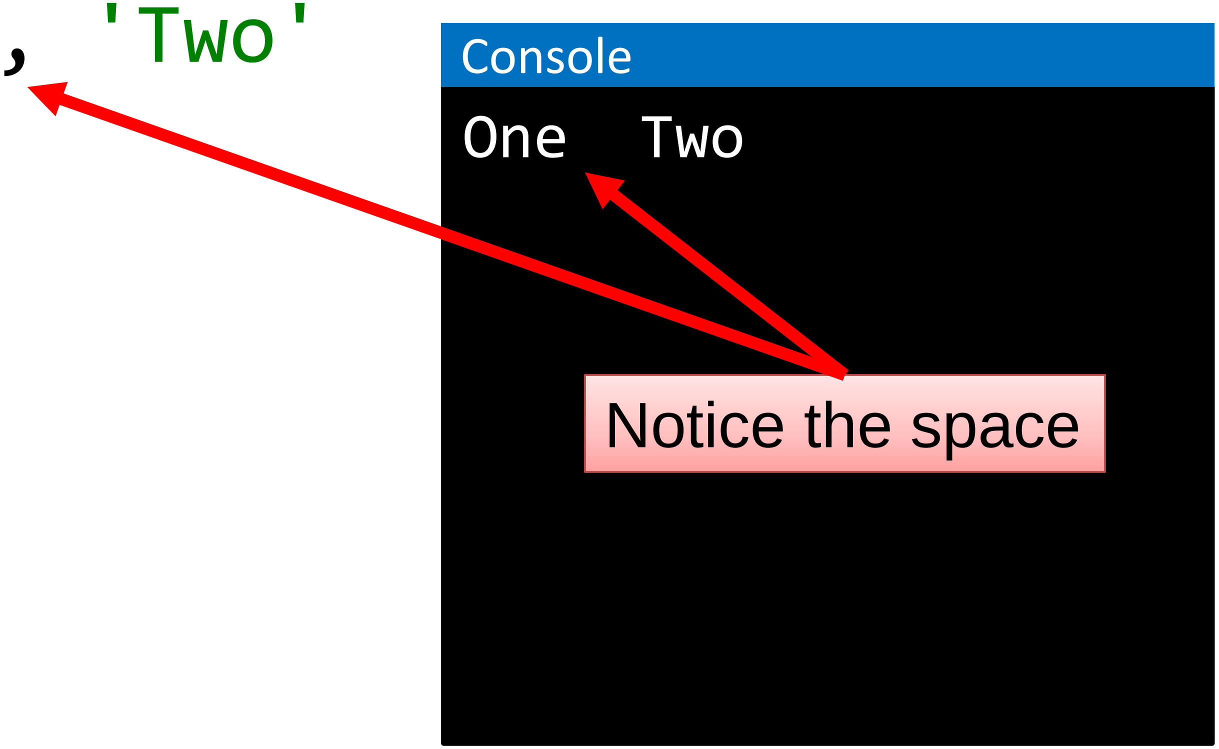
```
print 'One'
```

Console

```
One
```

print Examples

```
print 'One', 'Two'
```



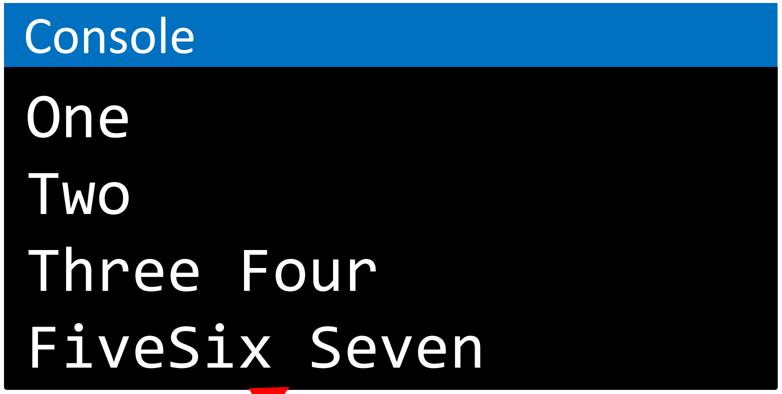
Console

```
One  Two
```

Notice the space

print Examples

```
print 'One'
print 'Two'
print 'Three',
print 'Four'
print 'Five' 'Six', 'Seven'
```



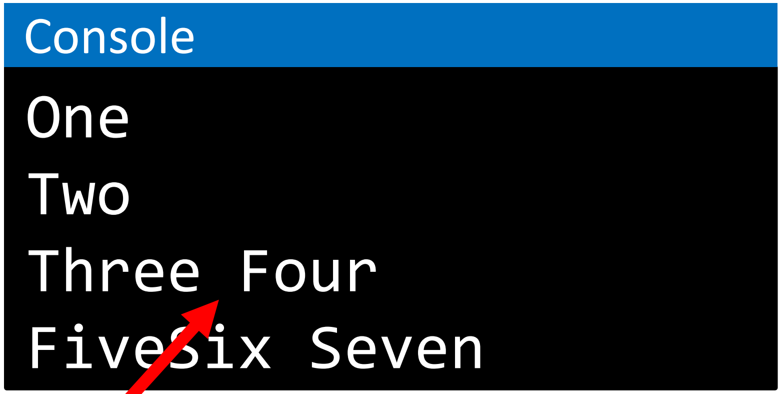
Console

```
One
Two
Three Four
FiveSix Seven
```

- Note that print goes to a new line by default
- A comma adds a space and can also specify staying on the same line

print Examples

```
print 'One'
print 'Two'
print 'Three',
print 'Four'
print 'Five' 'Six', 'Seven'
```

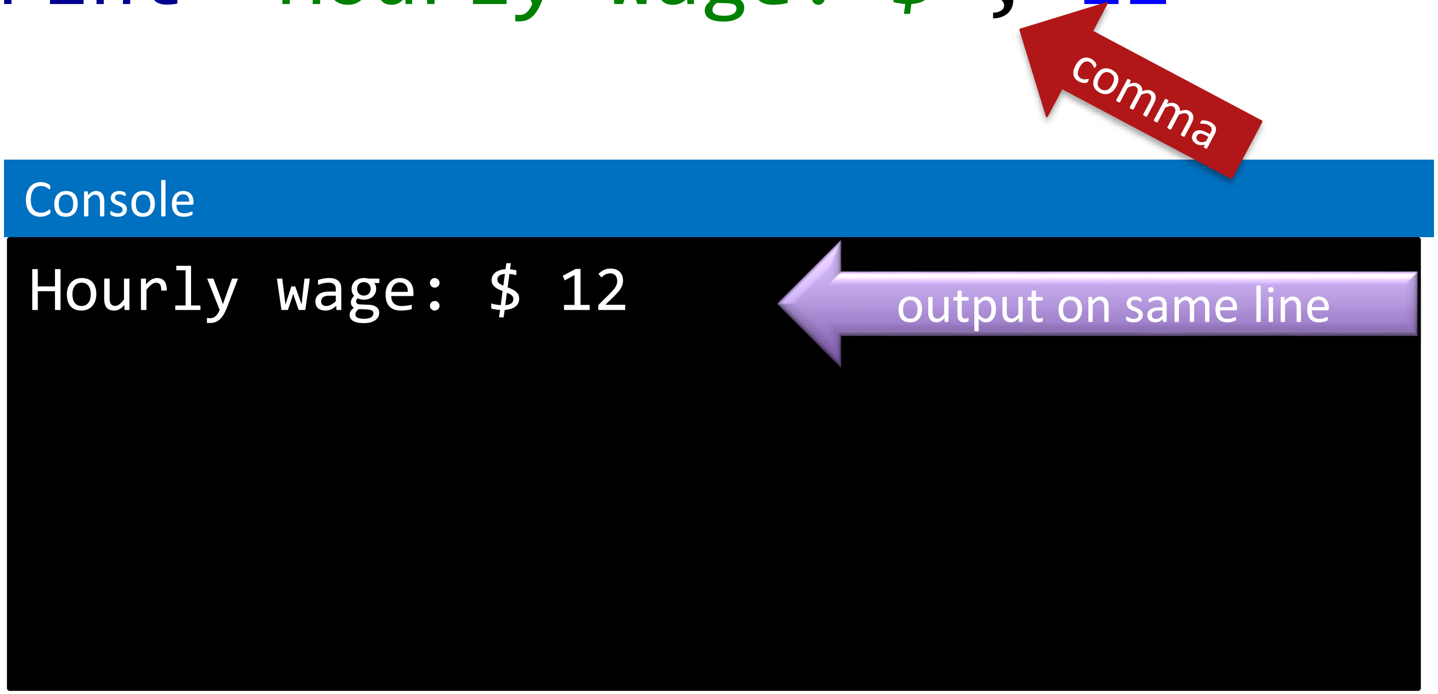


```
One
Two
Three Four
FiveSix Seven
```

- Note that print goes to a new line by default
- A comma adds a space and can also specify staying on the same line

Print multiple items
separate them with a ,

```
print 'Hourly wage: $', 12
```



comma

Console

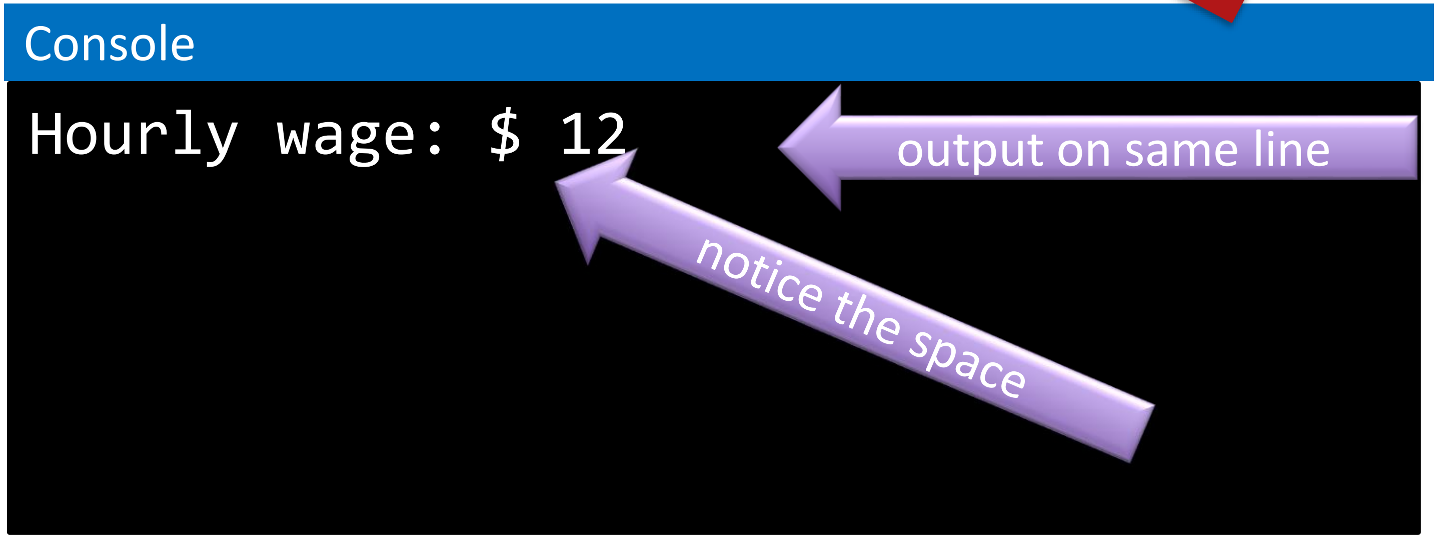
```
Hourly wage: $ 12
```

output on same line

Print multiple items
separate them with a ,

```
print 'Hourly wage: $', 12
```

comma



Console

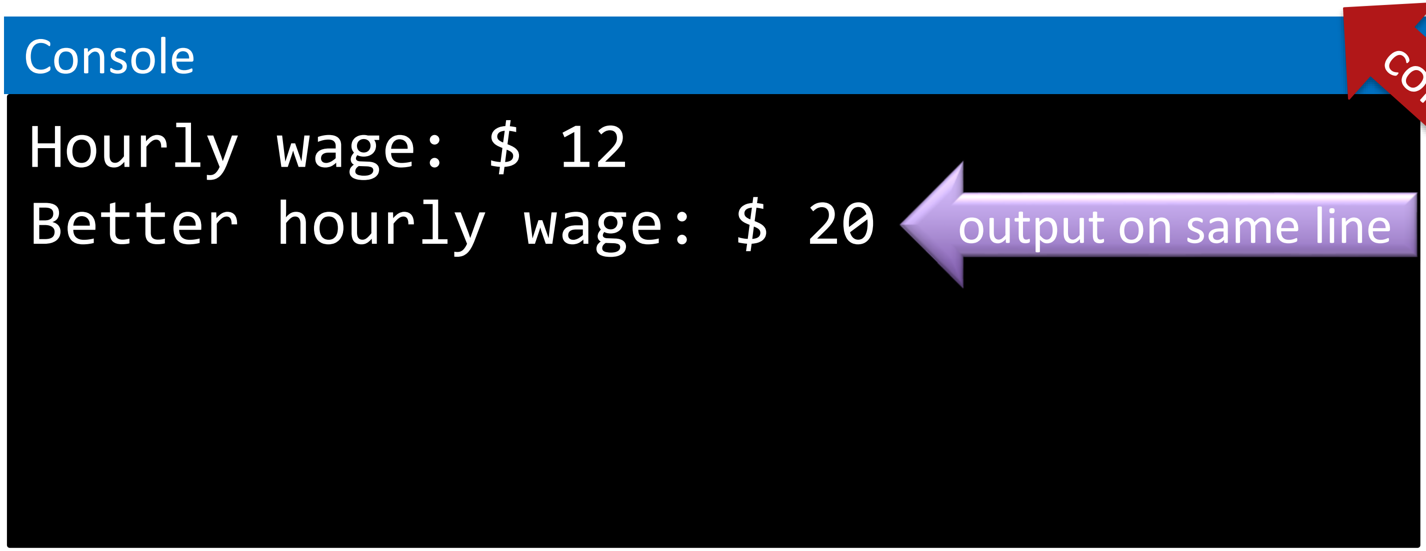
```
Hourly wage: $ 12
```

output on same line

notice the space

Print multiple items
separate them with a ,

```
print 'Hourly wage: $', 12
print 'Better hourly wage: $', 20
```



Console

```
Hourly wage: $ 12
Better hourly wage: $ 20
```

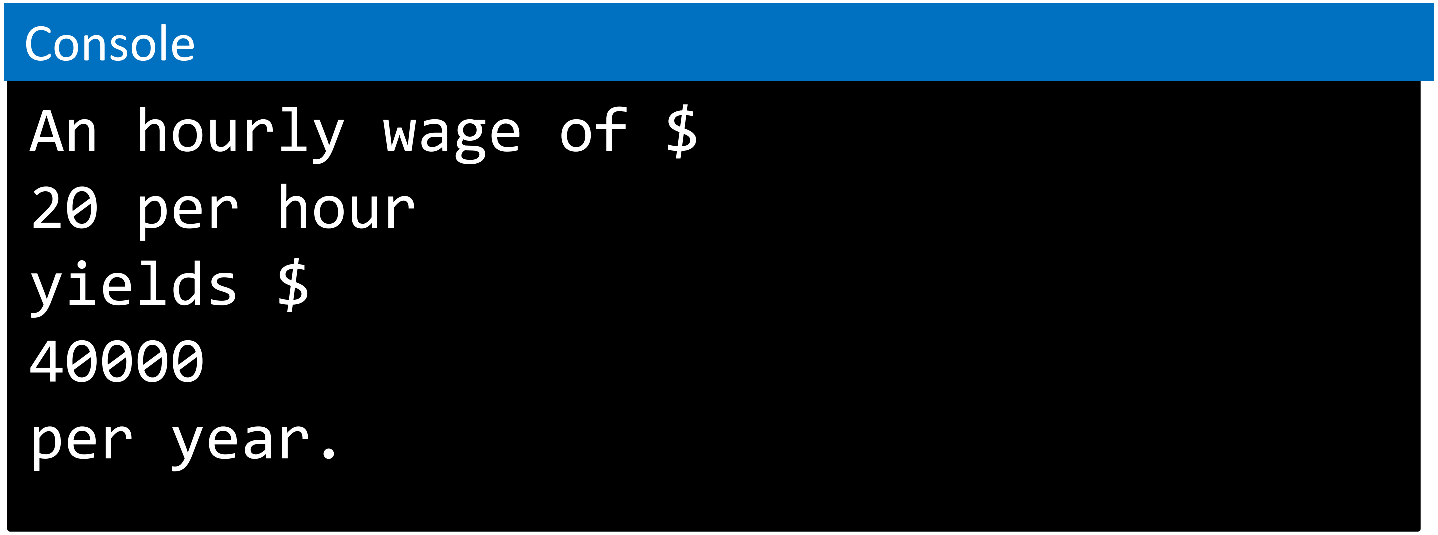
output on same line

comma

Print multiple items
each “print” outputs on its own line

```
hourlyWage = 20
print 'An hourly wage of $'
print hourlyWage, 'per hour'
print 'yields $'
print hourlyWage * 40 * 50
print 'per year.'
```

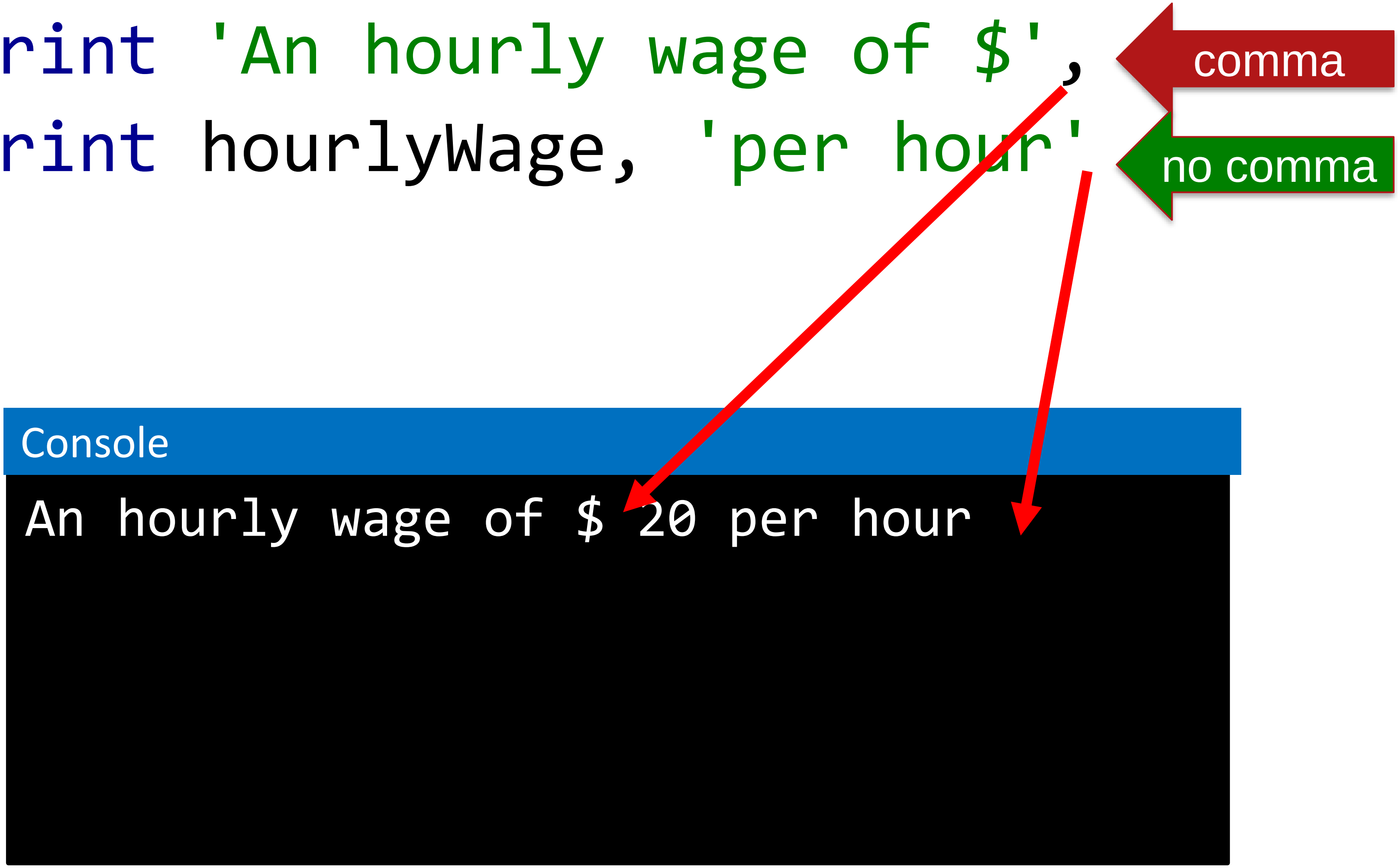
no comma
on
line ends



```
An hourly wage of $
20 per hour
yields $
40000
per year.
```

Print multiple items –
add , to suppress the newline

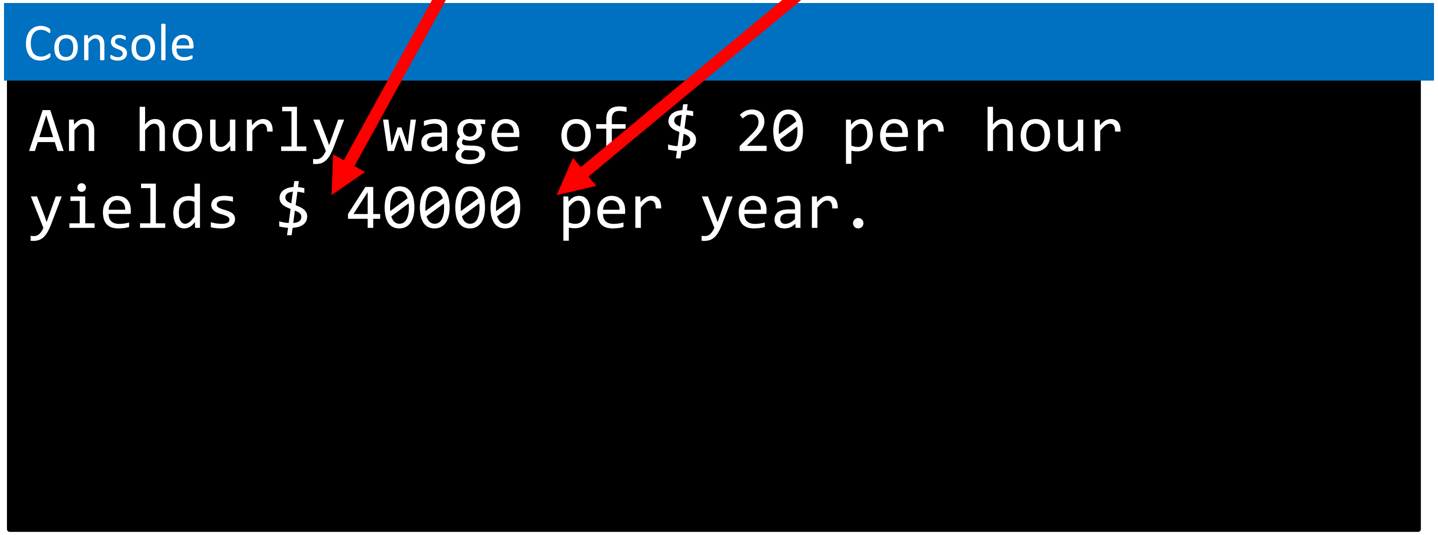
```
hourlyWage = 20
print 'An hourly wage of $',
print hourlyWage, 'per hour'
```



```
An hourly wage of $ 20 per hour
```

Print multiple items –
add , to suppress the newline

```
hourlyWage = 20
print 'An hourly wage of $',
print hourlyWage, 'per hour'
print 'yields $',
print hourlyWage * 40 * 50,
print 'per year.'
```



```
An hourly wage of $ 20 per hour
yields $ 40000 per year.
```

( ) as continuation symbol
one “print” multiple lines of code

```
hourlyWage = 20
print 'An hourly wage of $' + (
    str(hourlyWage) + 'per hour')
```

paren

paren

no print

Console

```
An hourly wage of $20 per hour
```

newline

( ) as continuation symbol
one “print” multiple lines of code

```
hourlyWage = 20
print 'An hourly wage of $' + (
     str(hourlyWage) + 'per hour')
```

cast to a string

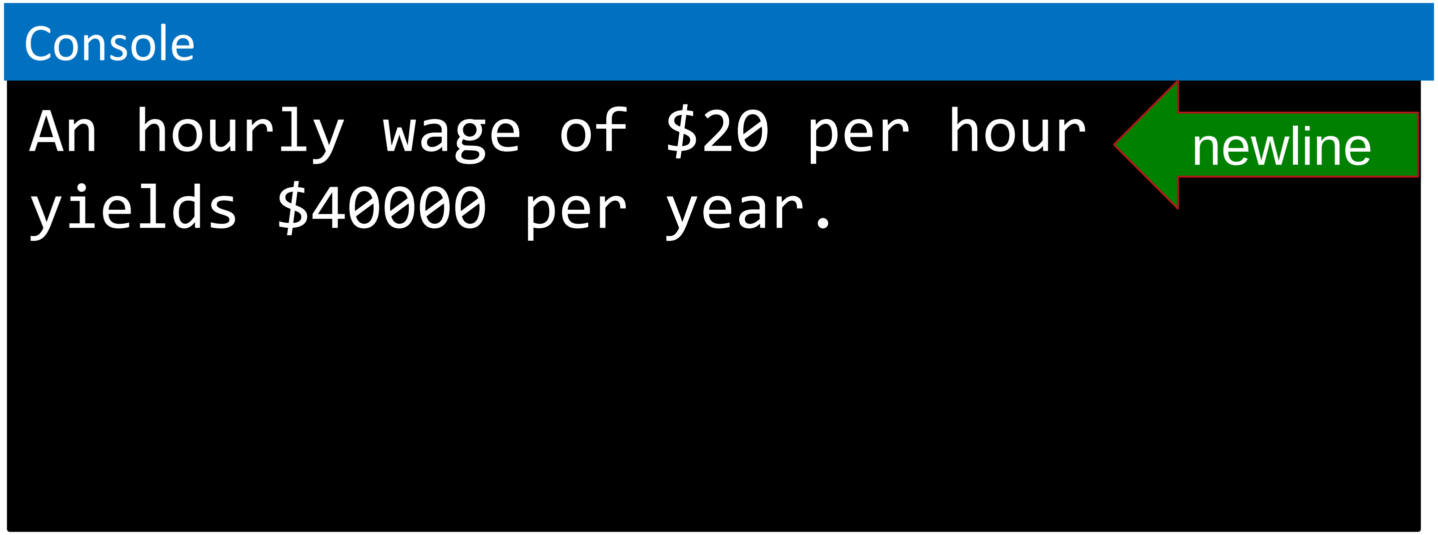
Console

```
An hourly wage of $20 per hour
```

newline

( ) as continuation symbol
one “print” multiple lines of code

```
hourlyWage = 20
print 'An hourly wage of $' + (
    str(hourlyWage) + 'per hour')
print 'yields $' + (
    str(hourlyWage * 40 * 50) +
```



Console

```
An hourly wage of $20 per hour
yields $40000 per year.
```

Comma and space on output

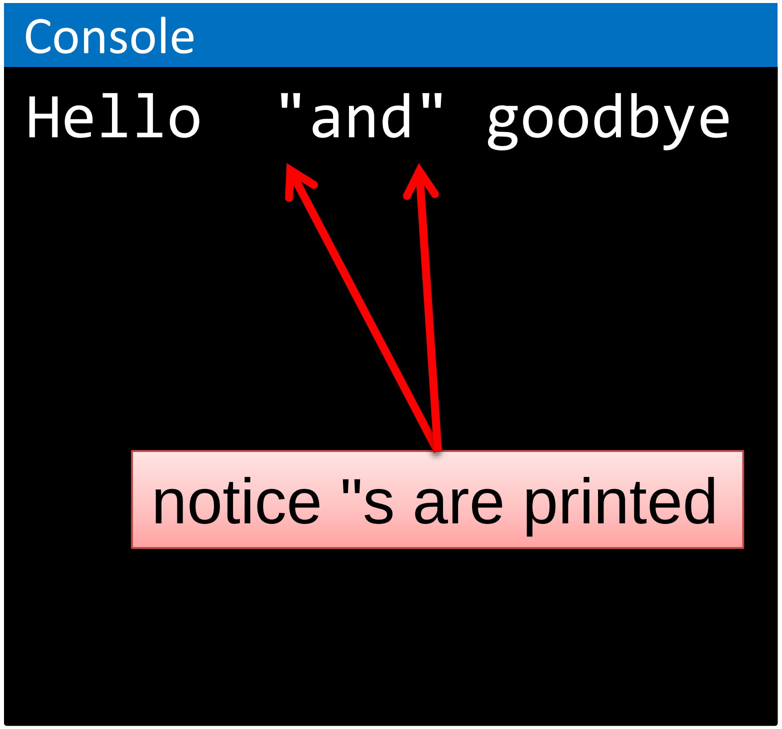
- Comma between items (outside of quotes):
  - Python inserts a blank space
- Comma at the end of a print statement:
  - Inserts a space, instead of a newline
- Don’t want blank space?
  - Use something other than comma
  - Such as **+** for concatenation

Special Output Characters used in printing

\n	new line
\t	tab
\b	backspace
\r	carriage return
\'	single quote
\"	double quote
\\	backslash

Example: escape char on "

```
print "Hello \"and\" goodbye"
```



```
Hello  "and" goodbye
```

notice "s are printed

Example: without using \

- You get the same results by starting and ending the string with single quotes
  - The double quotes on the inside do not mean "start" or "end" of the string, they're just another character

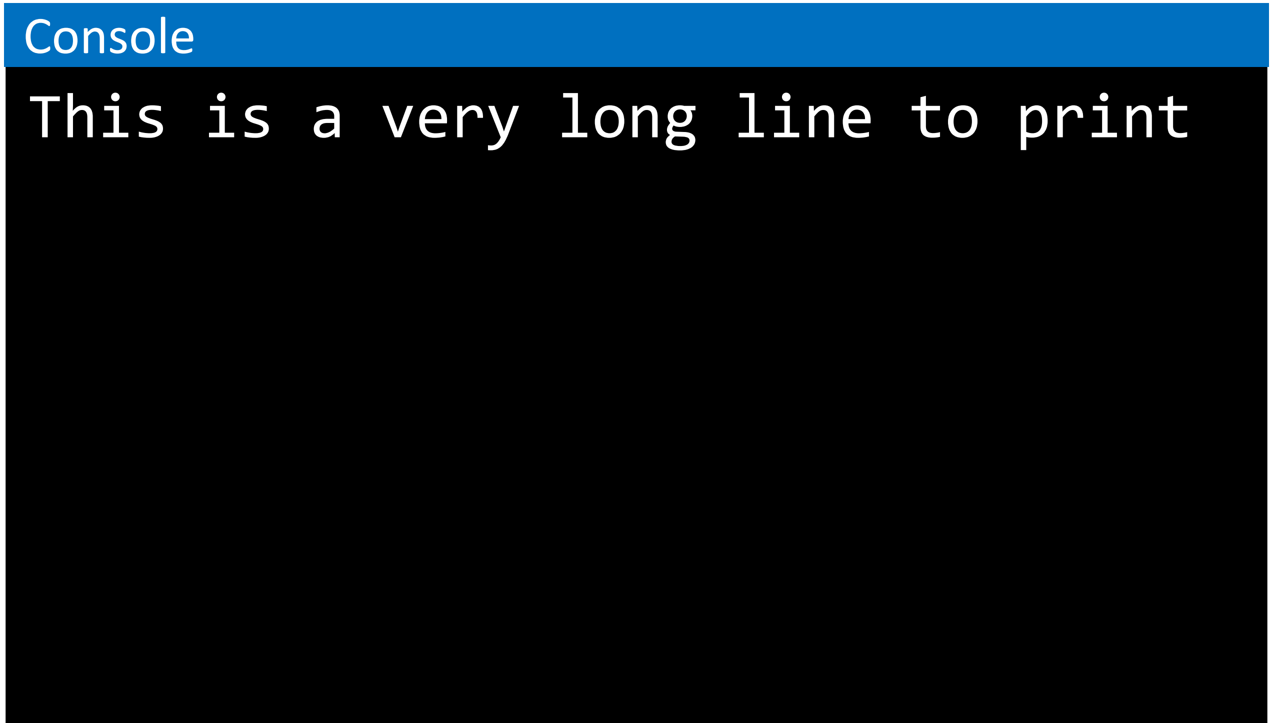
```
print 'Hello "and" goodbye'
```

  - Need a single quote inside? Use double outside:

```
print "It's raining again"
```

Example: long line

```
print 'This is a very long' + (
      ' line to print')
```



```
This is a very long line to print
```

i>Clicker #5

```
>>> print "EECS183" + "Lecture" + "Rocks!"
```

What prints?

```
A) EECS183
   Lecture
   Rocks!
B) EECS183 Lecture Rocks!
C) EECS183LectureRocks!
```

D) This is not valid Python

i>Clicker #5

```
>>> print "EECS183" + "Lecture" + "Rocks!"
```

What prints?

```
A) EECS183
   Lecture
   Rocks!
B) EECS183 Lecture Rocks!
```

[✓ C) `EECS183LectureRocks!`]

D) This is not valid Python

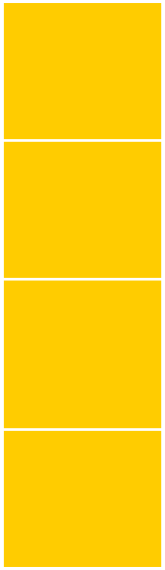
raw_input()

- Ignores leading and trailing white spaces
  - Does NOT store them as part of the string
- Stops reading when it hits `<enter>`
- Returns a `str` datatype

raw_input()

Execution

```
print 'Enter a saying: '

saying = raw_input()
```



Console

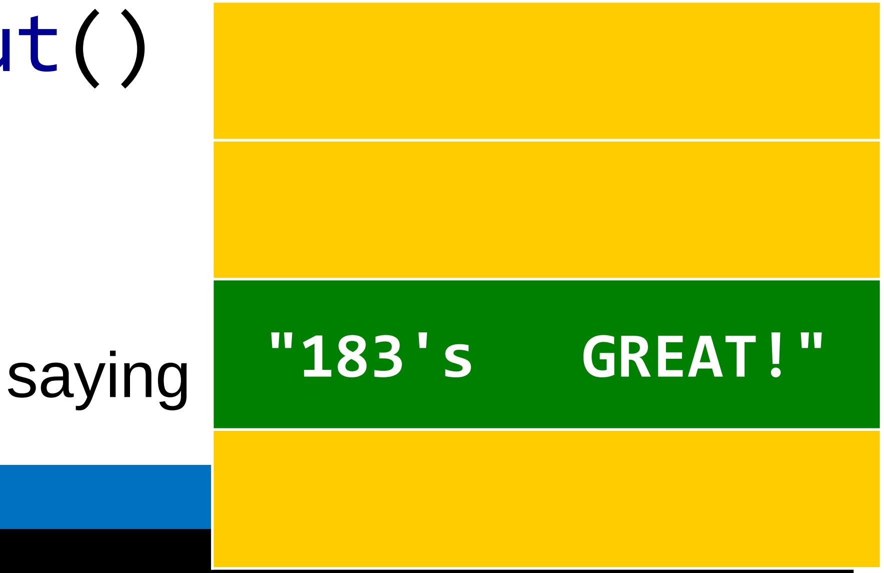
```
Enter a saying:

_
```

raw_input()

```
print 'Enter a saying: '
```

Execution

```
saying = raw_input()
```



Console

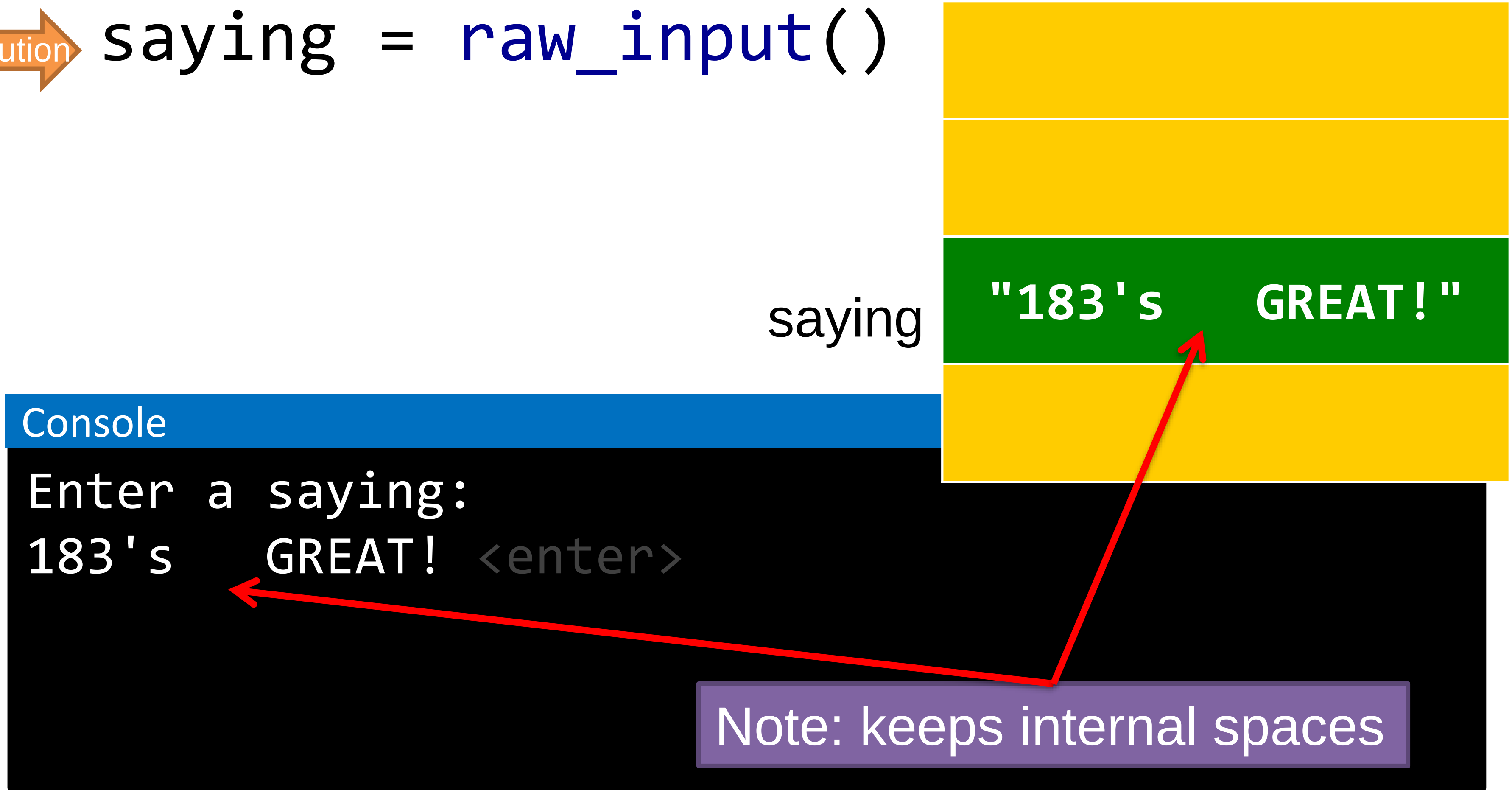
```
Enter a saying:
183's   GREAT! <enter>
```

raw_input()

```
print 'Enter a saying: '
```

Execution

```
saying = raw_input()
```



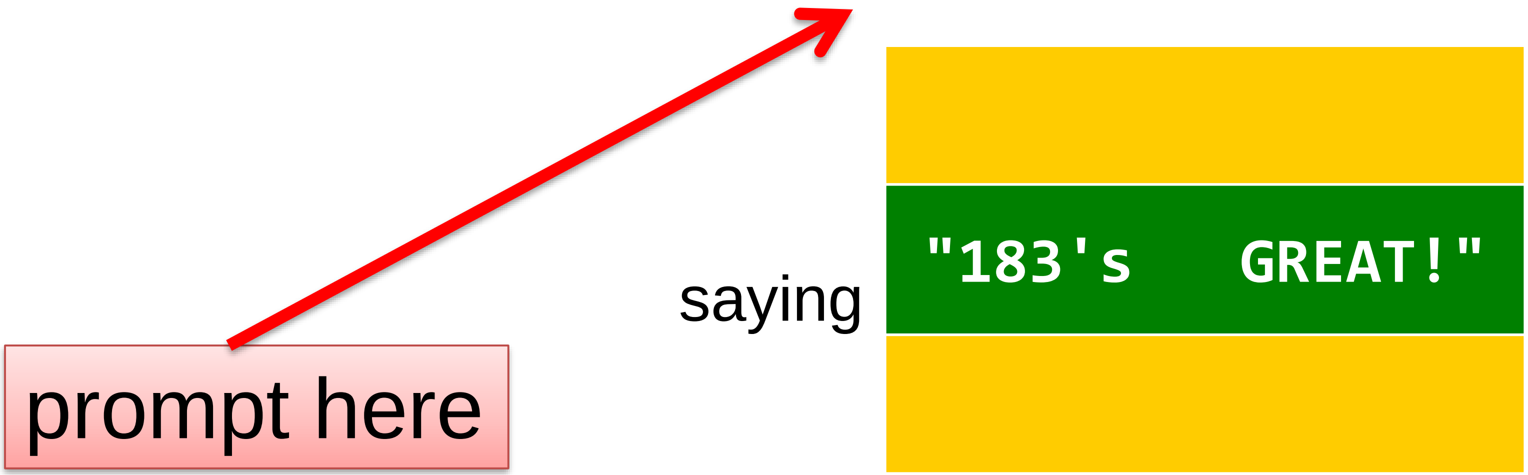
```
Enter a saying:
183's    GREAT! <enter>
```

Note: keeps internal spaces

raw_input() - remember

Execution

```
saying = raw_input('Enter a saying: ')
```



prompt here

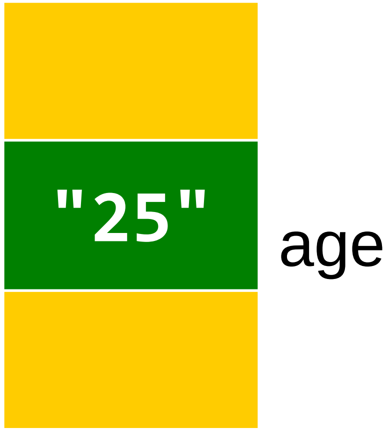
Console

```
Enter a saying:
183's    GREAT! <enter>
```

raw_input() - remember

Execution

```
age = raw_input('Enter your age: ')
```



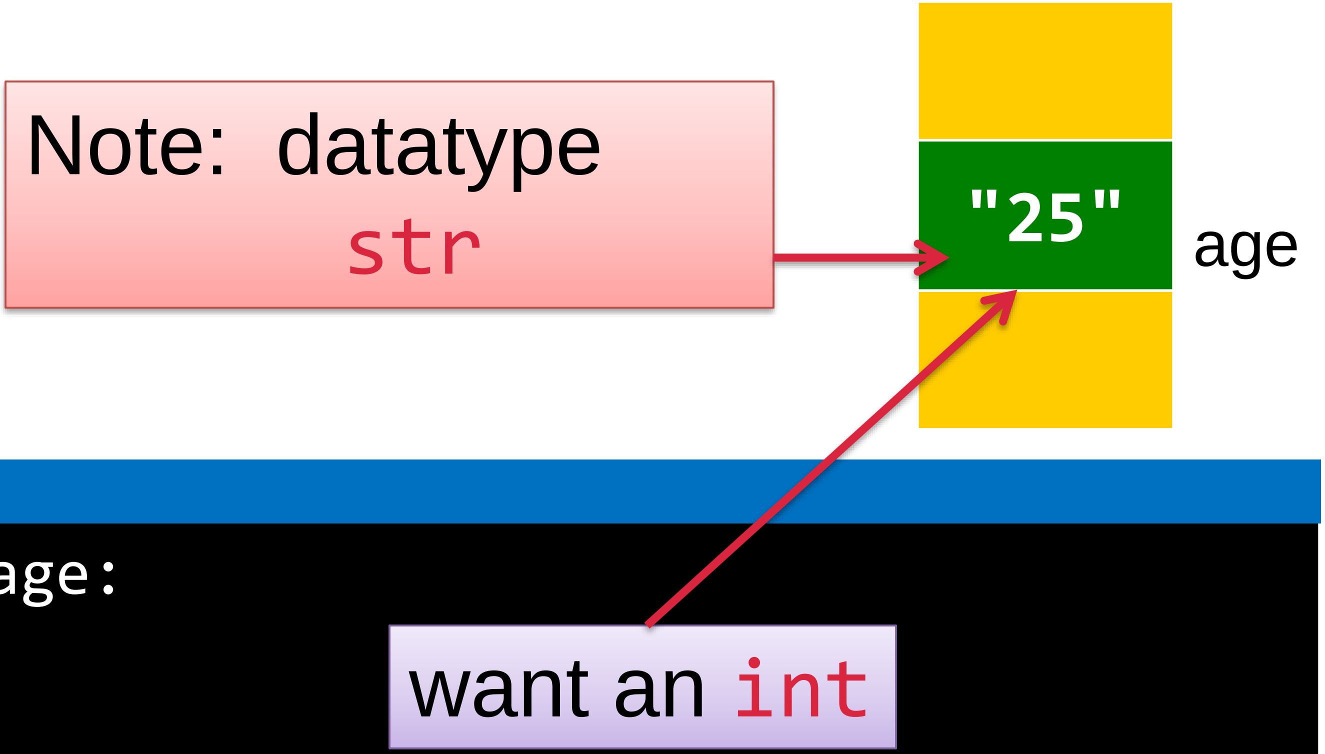
Console

```
Enter your age:
25<enter>
```

raw_input() - remember

Execution

```
age = raw_input('Enter your age: ')
```



Note: datatype
str

want an int

Console

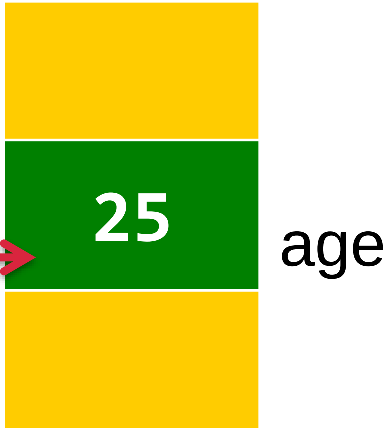
```
Enter your age:
25<enter>
```

raw_input()

Execution

```
age = int(raw_input('Enter your age: '))
```

Note: cast to int



Console

```
Enter your age:
25<enter>
```

Pythonic line continuation

```
length = int(raw_input('Enter length: '))

width = int(raw_input('Enter width: '))

print 'The area of the rectangle is:', (
    length * width )
```

Use a set of ( ) to indicate that more values are forthcoming for the above statement

Built-in Functions

`int()`	`raw_input()`	`abs()`
`float()`	`print ()`	`min()`
`bool()`	`ord()`	`max()`
	`chr()`	`round()`

For a full list, see:
http://docs.python.org/2/library/functions.html

Built-in Functions
Cast functions

`int()`

`float()`

`bool()`

`str()`

```
print 'Enter the length:',
length = int(raw_input())
```

takes the `str` returned by `raw_input()` and converts it to an `int`

Built-in Functions
Cast functions

```
int()                bool()
float()              str()
```

```
print 'Enter the length:',
length = float(raw_input())
```

takes the `str` returned by `raw_input()` and converts it to a `float`

Built-in Functions
abs

- returns the absolute value of a value

```
print abs(4.2)
4.2

print abs(-5)
5
```

Built-in Functions
min & max

- min returns smallest of arguments

```
print min(3, 5)
3

print min(3, 2, 7, 10, 1)
1
```

Built-in Functions
min & max

- `min` returns smallest of arguments
- `max` returns largest of arguments

```
print max(3, 5)
5

print max(3, 2, 7, 10, 1)
10
```

import math

- math.pi
- math.e
- math.ceil(x)
- math.floor(x)
- math.fabs(x)
- math.pow(x,y)
- math.sqrt(x)

```
# Example
import math
print math.pi
x = math.sqrt(42)
print x
```

import math

- Basic Trig functions
  - `math.sin(x)`
  - `math.cos(x)`
  - `math.tan(x)`
- Arc Trig functions
  - `math.asin(x)`
  - `math.acos(x)`
  - `math.atan(x)`

all **radian** based

Conversions:

```
math.degrees(rad)
math.radians(deg)
```

import math
Examples:

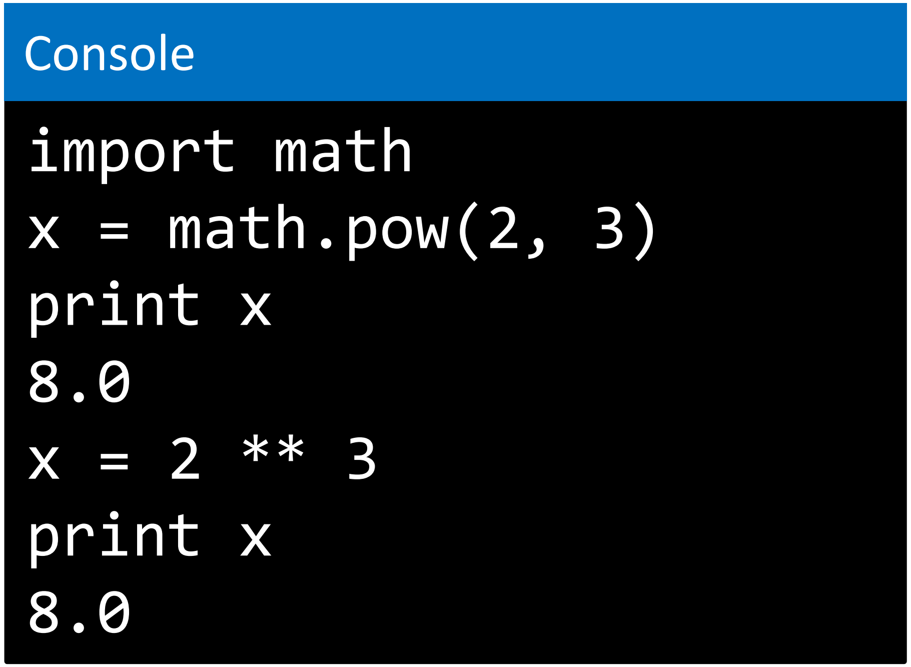
- math.pi
- math.e
- math.ceil(x)
- math.floor(x)
- math.fabs(x)
- math.pow(x,y)
- math.sqrt(x)

Console

```
import math
x = math.sqrt(42)
print x
6.48074069841
```

import math
Examples:

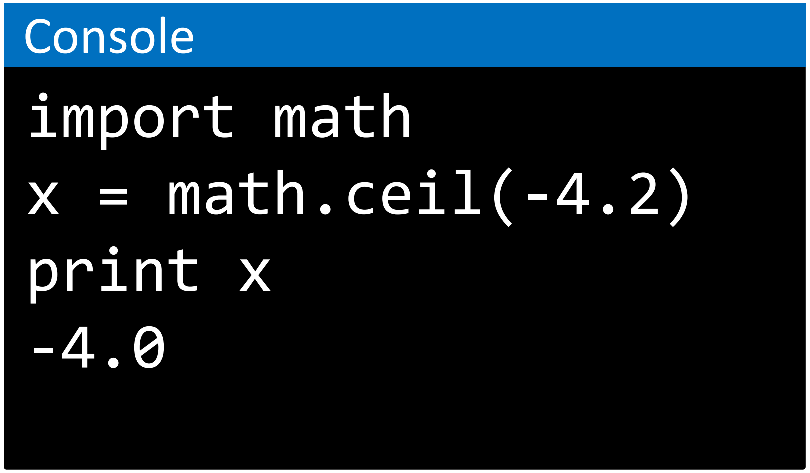
- `math.pi`
- `math.e`
- `math.ceil(x)`
- `math.floor(x)`
- `math.fabs(x)`
- `math.pow(x,y)`
- `math.sqrt(x)`



```
import math
x = math.pow(2, 3)
print x
8.0
x = 2 ** 3
print x
8.0
```

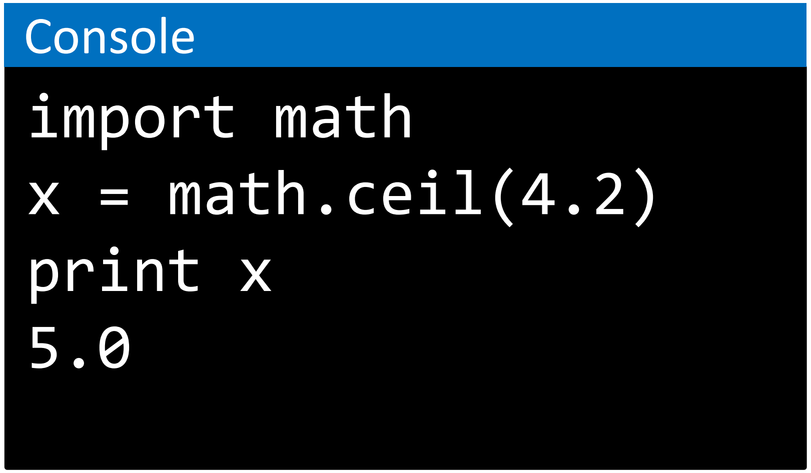
import math
Examples:

```
# returns rounded up value of x
math.ceil(x)
# returns rounded down value of x
math.floor(x)
```



Console

```
import math
x = math.ceil(-4.2)
print x
-4.0
```

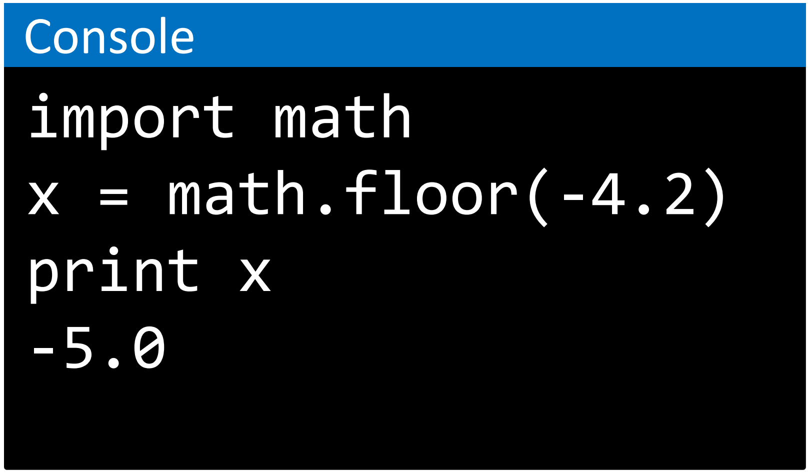


Console

```
import math
x = math.ceil(4.2)
print x
5.0
```

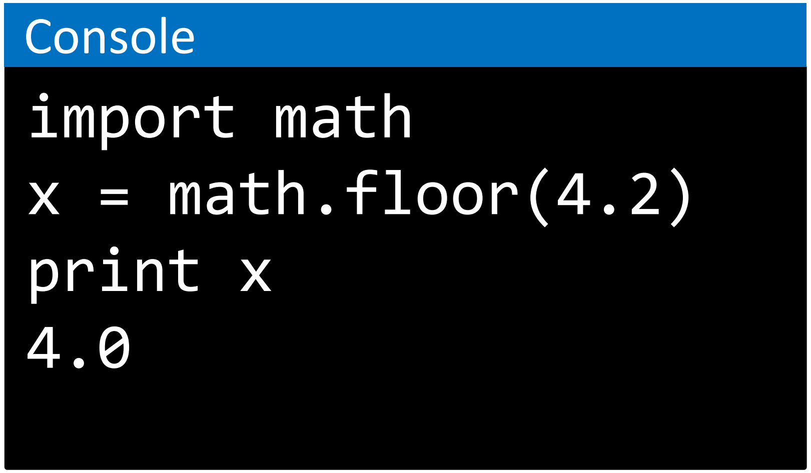
import math
Examples:

```
# returns rounded up value of x
math.ceil(x)
# returns rounded down value of x
math.floor(x)
```



Console

```
import math
x = math.floor(-4.2)
print x
-5.0
```



Console

```
import math
x = math.floor(4.2)
print x
4.0
```

Making comparisons

- Python supports all of the same relational operators:

```
== != < <= > >=
```

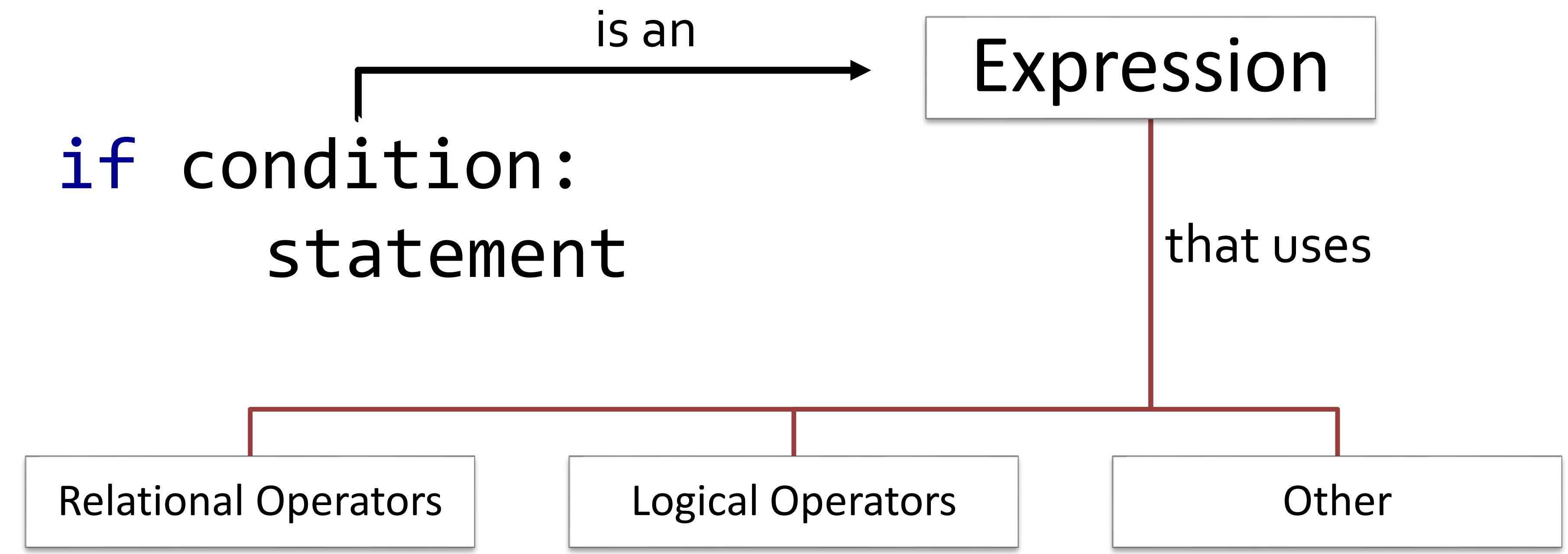
- The logical operators are typed out:

```
and or not
```

- Parentheses are not required
  - (Python programmers don’t use them)

```
if x > 0:
```

Conditions



is an

Expression

```
if condition:
       statement
```

that uses

Relational Operators

Logical Operators

Other

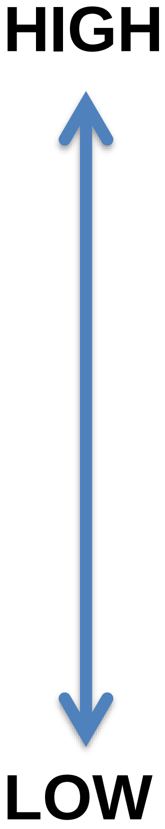
==	is equal to
!=	is not equal to
<	is smaller than
<=	is smaller than or equal to
>	is greater than
>=	is greater than or equal to

and
or
not

( )
Mathematical operators
others

Precedence Rules *Recap*

OPERATOR	***ASSOCIATIVITY***
( )	left to right
** (exponentiation)	right to left
+x -x cast	right to left
* / %	left to right
+ - (add, subtract)	left to right
< <= > >= == !=	left to right
not	left to right
and	left to right
or	left to right
=	right to left



HIGH

LOW

Problem Python *almost* avoids

- Always be careful to use `==` in an `if`, not `=`
- In C++ you get unexpected results
- In Python you usually get an error

```
x = 3
if x = 4:
SyntaxError: invalid syntax
```

Why “almost”?

- In Python you can still make the = mistake with boolean variables:

```
done = False
if done = True:
    print 'Done!'
```

- This code always displays Done!

Avoid this problem!

- **NEVER** compare `== True` or `== False`
- Use the boolean, and use `not` if necessary

```
if done:
```

```
if not done:
```

The scope of if

```
someBool = True
if someBool:
    print 'This is in the if scope.'
    print 'This is ALSO in the if scope.'
    print 'Even this is in the if scope.'
print 'But this is NOT in the if scope.'
```

The scope of if is set by indent

Look! No Braces!!!

```
someBool = True
if someBool:
    print 'This is in the if scope.'
    print 'This is ALSO in the if scope.'
    print 'Even this is in the if scope.'
print 'But this is NOT in the if scope.'
```

The indent sets the scope of the if!

Discount books example

```
DISCOUNT = 0.30

print 'Enter list price of book: ',
price = float(raw_input())
print 'Is it used? Y or N: ',
usedCode = raw_input()

if usedCode == 'Y' or usedCode == 'y':
    print 'Applying used discount'
    price = price - (DISCOUNT * price)

print 'Selling price $', price
```

What about else?

```
DISCOUNT = 0.30

print 'Enter list price of book: ',
price = float(raw_input())
print 'Is it used? Y or N: ',
usedCode = raw_input()
```

```
if usedCode == 'Y' or usedCode == 'y':
    print 'Applying used discount'
    price = price - (DISCOUNT * price)
else:
    print 'Full price'

print 'Selling price $', price
```

Using “else if” in Python: elif

```
score = float(raw_input('Enter score: '))

if score >= 90:
    print 'Pass with an A grade'
elif score >= 80:
    print 'Pass with a B grade'
elif score >= 70:
    print 'Pass with a C grade'
else:
    print 'Not passing'
```

Multiple comparisons, same variable

- Suppose we wanted to check whether a number was in a range, like a test score
- In C++ you had to have two clauses and link them with &&
- The same thing can be done in Python:

```
if 0 <= score and score <= 100:
```

- However, Python has a shortcut that does not work in C++:

```
if 0 <= score <= 100:
```