ECE-203 – Programming for Engineers

Contact

Dr. James Shackleford
shack@drexel.edu
Bossone 211

Office Hours: 3 – 4 pm (Tuesday)
Course Website: http://learn.dcollege.net

Textbook

Think Python
by Allen Downey
O'Reilly Press, 2015
ISBN-13: 978-1449330729
(Freely available in PDF format, check course website)

Grading

- 10% In-lab Programming Assignments
- 10% Take-Home Programming Assignments
- 35% Mid-term Exam
- 45% Final Exam

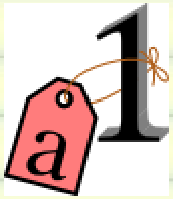
A Few Notes on Representations

In the previous lecture, we learned about *objects* and how to bind *names* to objects

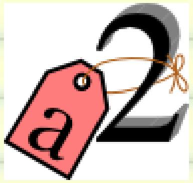
>> Name binding is kinda unique to Python
>> You can later define your own custom objects (later...)

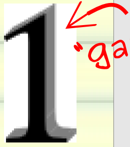
Python

```
a = 1
```



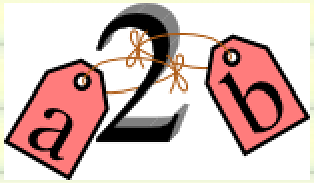
```
a = 2
```





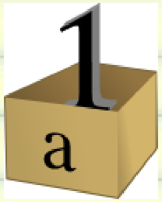
Gets "garbage collected" (no bindings)

```
b = a
```

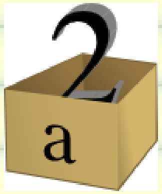


Most Other Languages

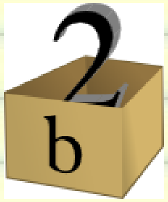
```
int a = 1;
```

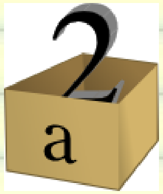


```
int a = 2;
```



```
int b = a;
```





A Few Notes on Representations

Everything in Python is an object!

```
>> a = 2                    # a binds to an integer (int) object  :  2
>> b = 13                   # b binds to an integer (int) object  :  13
>> c = 9.0                  # c binds to a real number (float) object  : 9.0
>> d = b/a                  # d binds to int/int => (int) object  : 13/2 = 6
>> e = c/a                  # e binds to float/int => (float) object  : 9.0/2 = 4.5
>> f = 'b/a=%g' % (b/a)     # f binds to a (str) object  : ‘b/a=6’
```

A Few Notes on Representations

Converting Objects to Another Type

```
>> a = 2                  # a binds to an (int) containing: 2
>> b = float(a)           # b binds to a new (float) object containing: 2.0
>> c = 6.8                # c binds to a (float) containing: 6.8
>> d = int(c)             # d binds to a new (int) object containing: 6
>> d = round(c)           # d binds to a new (float) containing: 7.0
>> d = int(round(c))      # d binds to a new (int) containing: 7
>> d = str(c)             # c binds to a new (str) containing: “6.8”
>> e = “45.23”            # e binds to a (str) containing '45.23'
>> f = float(e)           # f binds to a new (float) containing: 45.23
```

Sequence Attributes of Strings

A Bit More on Strings

Strings are one of the three *Sequence Types* in Python

(the other two are tuples and lists)

★ **Strings are immutable – you can't change strings**

(but you can create new strings from other strings)

Sequence Attributes of Strings

A Bit More on Strings

Strings are one of the three *Sequence Types* in Python

(the other two are tuples and lists)

- ★ **Strings are immutable – you can't change strings**

  (but you can create new strings from other strings)

- ★ **Strings are defined using quotes – (“, ', or “““)**

```
>> my_string = “Hello World”       # This and
>> my_string = 'Hello World'       #       ...this are the same

>> my_string = “““This is a multi-line
string that uses triple quotes”””
```

Sequence Attributes of Strings

A Bit More on Strings

Strings are one of the three *Sequence Types* in Python

(the other two are tuples and lists)

★ **Strings are immutable – you can't change strings**

(but you can create new strings from other strings)

★ **Strings are defined using quotes – (“, ', or “““)**

```
>> my_string = “Hello World”      # This and
>> my_string = 'Hello World'      #       ...this are the same

>> my_string = “““This is a multi-line
string that uses triple quotes”””
```

★ **The different quotes allow you to actually use quotes in your strings!**

```
>> my_string = 'Shackleford said, “Learn Python”'

>> print my_string
Shackleford said, “Learn Python”
```

Sequence Attributes of Strings

A Bit More on Strings

Strings are one of the three *Sequence Types* in Python

(the other two are tuples and lists)

★ All Sequence Types can be indexed – this includes Strings

```
>> my_string = “Hello World”
>> first_character = my_string[0]
>> print first_character
H

>> print my_string[3]
l
```

(you played with this a bit in lab)

Sequence Attributes of Strings

A Bit More on Strings

Strings are one of the three *Sequence Types* in Python

(the other two are tuples and lists)

★ **All Sequence Types can be indexed – this includes Strings**

```
>> my_string = “Hello World”
>> first_character = my_string[0]
>> print first_character
H

>> print my_string[3]
l
```

(you played with this a bit in lab)

★ **Negative indices “wrap around” to the end and “go backwards”**

```
>> my_string = “Hello World”
>> last_character = my_string[-1]
>> print last_character
d

>> print my_string[-3]
r
```

Sequence Attributes of Strings

A Bit More on Strings

Strings are one of the three *Sequence Types* in Python

(the other two are tuples and lists)

★ **All Sequence Types can be sliced – this includes Strings**

This creates a new string object containing a subset of a string

```
>> my_string = "Hello World"

>> my_string[1:4]
'ell'
```

Sequence Attributes of Strings

A Bit More on Strings

Strings are one of the three *Sequence Types* in Python

(the other two are tuples and lists)

★ All Sequence Types can be sliced – this includes Strings

This creates a new string object containing a subset of a string

```
>> my_string = “Hello World”

>> my_string[1:4]
'ell'

>> my_string[1:-1]
'ello Worl'
```

Sequence Attributes of Strings

A Bit More on Strings

Strings are one of the three *Sequence Types* in Python

(the other two are tuples and lists)

★ All Sequence Types can be sliced – this includes Strings

This creates a new string object containing a subset of a string

```
>> my_string = “Hello World”

>> my_string[1:4]
'ell'

>> my_string[1:-1]
'ello Worl'

>> my_string[:5]
'Hello'
```

Sequence Attributes of Strings

A Bit More on Strings

Strings are one of the three *Sequence Types* in Python

(the other two are tuples and lists)

★ All Sequence Types can be sliced – this includes Strings

This creates a new string object containing a subset of a string

```
>> my_string = “Hello World”

>> my_string[1:4]
'ell'

>> my_string[1:-1]
'ello Worl'

>> my_string[:5]
'Hello'

>> my_string[6:]
'World'
```

Sequence Attributes of Strings

A Bit More on Strings

Strings are one of the three *Sequence Types* in Python

(the other two are tuples and lists)

★ **All Sequence Types can be sliced – this includes Strings**

This creates a new string object containing a subset of a string

```
>> my_string = “Hello World”

>> my_string[1:4]
'ell'

>> my_string[1:-1]
'ello Worl'

>> my_string[:5]
'Hello'

>> my_string[6:]
'World'

>> my_string[:]
'Hello World'
```

More Attributes of Strings

A Bit More on Strings

Strings are one of the three *Sequence Types* in Python

(the other two are tuples and lists)

★ Subsequence Matching using the `in` operator

```
>>> my_string = “Hello World”

>>> 'Wor' in my_string
True
```

More Attributes of Strings

A Bit More on Strings

Strings are one of the three *Sequence Types* in Python

(the other two are tuples and lists)

★ Subsequence Matching using the `in` operator

```
>>> my_string = “Hello World”

>>> 'Wor' in my_string
True

>>> 'Wr' in my_string
False
```

More Attributes of Strings

A Bit More on Strings

Strings are one of the three *Sequence Types* in Python

(the other two are tuples and lists)

★ Subsequence Matching using the `in` operator

```
>>> my_string = “Hello World”

>>> 'Wor' in my_string
True

>>> 'Wr' in my_string
False

>>> if 'Hello' in my_string:
...     print 'Great Success!'
...
Great Success!
```

More Attributes of Strings

A Bit More on Strings

Strings are one of the three *Sequence Types* in Python

(the other two are tuples and lists)

★ **Strings can be concatenated**

```
>>> var1 = “Hello” + “ ” + “World”

>>> print var1
Hello World!
```

More Attributes of Strings

A Bit More on Strings

Like everything in Python, Strings are *objects*

THIS MEANS THEY HAVE METHODS

★ **Strings have lots of powerful methods!**

```
>>> foo = “i am a string!”
>>> foo.upper()
'I AM A STRING!'
```

https://docs.python.org/2/library/stdtypes.html#string-methods

More Attributes of Strings

A Bit More on Strings

Like everything in Python, Strings are *objects*

THIS MEANS THEY HAVE METHODS

★ **Strings have lots of powerful methods!**

```
>>> foo = “i am a string!”
>>> foo.upper()
'I AM A STRING!'

>> bar = “this is also valid!”.upper()
>> print bar
THIS IS ALSO VALID!
```

https://docs.python.org/2/library/stdtypes.html#string-methods

More Attributes of Strings

A Bit More on Strings

Like everything in Python, Strings are *objects*

THIS MEANS THEY HAVE METHODS

★ **Strings have lots of powerful methods!**

```
>>> foo = “i am a string!”
>>> foo.upper()
'I AM A STRING!'

>> bar = “this is also valid!”.upper()
>> print bar
THIS IS ALSO VALID!

>>> 'chapter 3: python makes programming fun'.title()
'Chapter 3: Python Makes Programming Fun'
```

https://docs.python.org/2/library/stdtypes.html#string-methods

More Attributes of Strings

A Bit More on Strings

Like everything in Python, Strings are *objects*

THIS MEANS THEY HAVE METHODS

★ **Strings have lots of powerful methods!**

```
>>> foo = “i am a string!”
>>> foo.upper()
'I AM A STRING!'

>> bar = “this is also valid!”.upper()
>> print bar
THIS IS ALSO VALID!

>>> 'chapter 3: python makes programming fun'.title()
'Chapter 3: Python Makes Programming Fun'

>>> “data1,label,data2,foo,bar”.split(“,”)
['data1', 'label', 'data2', 'foo', 'bar']
```

https://docs.python.org/2/library/stdtypes.html#string-methods

More Attributes of Strings

A Bit More on Strings

Like everything in Python, Strings are *objects*

THIS MEANS THEY HAVE METHODS

★ Strings have lots of powerful methods!

```
>>> foo = “i am a string!”
>>> foo.upper()
'I AM A STRING!'

>> bar = “this is also valid!”.upper()
>> print bar
THIS IS ALSO VALID!

>>> 'chapter 3: python makes programming fun'.title()
'Chapter 3: Python Makes Programming Fun'

>>> “data1,label,data2,foo,bar”.split(“,”)
['data1', 'label', 'data2', 'foo', 'bar']

>>> “-”.join( [“join”, “a”, “list”, “of”, “strings”] )
'join-a-list-of-strings'
```

https://docs.python.org/2/library/stdtypes.html#string-methods

More Attributes of Strings

A Bit More on Strings

Finally, Strings have the Formatting Operator %

Used to build a string by “filling in the blanks” with a single value, tuple, or dictionary

```
>>> speed = 10
>>> fuel = 5.23
>>> color = “blue”

>>> foo = “speed = %i” % speed
>>> print foo
'speed = 10'
```

For a table of conversion types (i.e. `%i`, `%f`, `%s`, etc) visit:

https://docs.python.org/2/library/stdtypes.html#string-formatting-operations

More Attributes of Strings

A Bit More on Strings

Finally, Strings have the Formatting Operator %

Used to build a string by “filling in the blanks” with a single value, tuple, or dictionary

```
>>> speed = 10
>>> fuel = 5.23
>>> color = “blue”

>>> foo = “speed = %i” % speed
>>> print foo
'speed = 10'

>> bar = “speed = %i, fuel = %f, and color is %s” % (speed, fuel, color)
>> print bar
'speed = 10, fuel = 5.230000, and color is blue'
```

For a table of conversion types (i.e. `%i`, `%f`, `%s`, etc) visit:

https://docs.python.org/2/library/stdtypes.html#string-formatting-operations

More Attributes of Strings

A Bit More on Strings

Finally, Strings have the Formatting Operator %

Used to build a string by “filling in the blanks” with a single value, tuple, or dictionary

```
>>> speed = 10
>>> fuel = 5.23
>>> color = “blue”

>>> foo = “speed = %i” % speed
>>> print foo
'speed = 10'

>> bar = “speed = %i, fuel = %f, and color is %s” % (speed, fuel, color)
>> print bar
'speed = 10, fuel = 5.230000, and color is blue'

>> baz = “speed = %i, fuel = %.2f, and color is %s” % (speed, fuel, color)
>> print baz
'speed = 10, fuel = 5.23, and color is blue'
```

For a table of conversion types (i.e. `%i`, `%f`, `%s`, etc) visit:

https://docs.python.org/2/library/stdtypes.html#string-formatting-operations

Course Philosophy

whew… alright

(let's remember why we are here)

GOAL 2

Solve numerical problems
...algorithmically

Focus on simulation, numerical methods, and heuristic methods of problem solving.

Solving Numerical Problems

Start with something simple

The constant acceleration problem

$$d(t) = v_0 t + \frac{1}{2} a t^2$$

$d(t)$	– distance at time t
v_0	– object's initial velocity @ $t = 0$
a	– object's constant acceleration

Solving Numerical Problems

Start with something simple

The constant acceleration problem

$$d(t) = v_0 t + \frac{1}{2} a t^2$$

$d(t)$ – distance at time t

v_0 – object's initial velocity @ $t = 0$

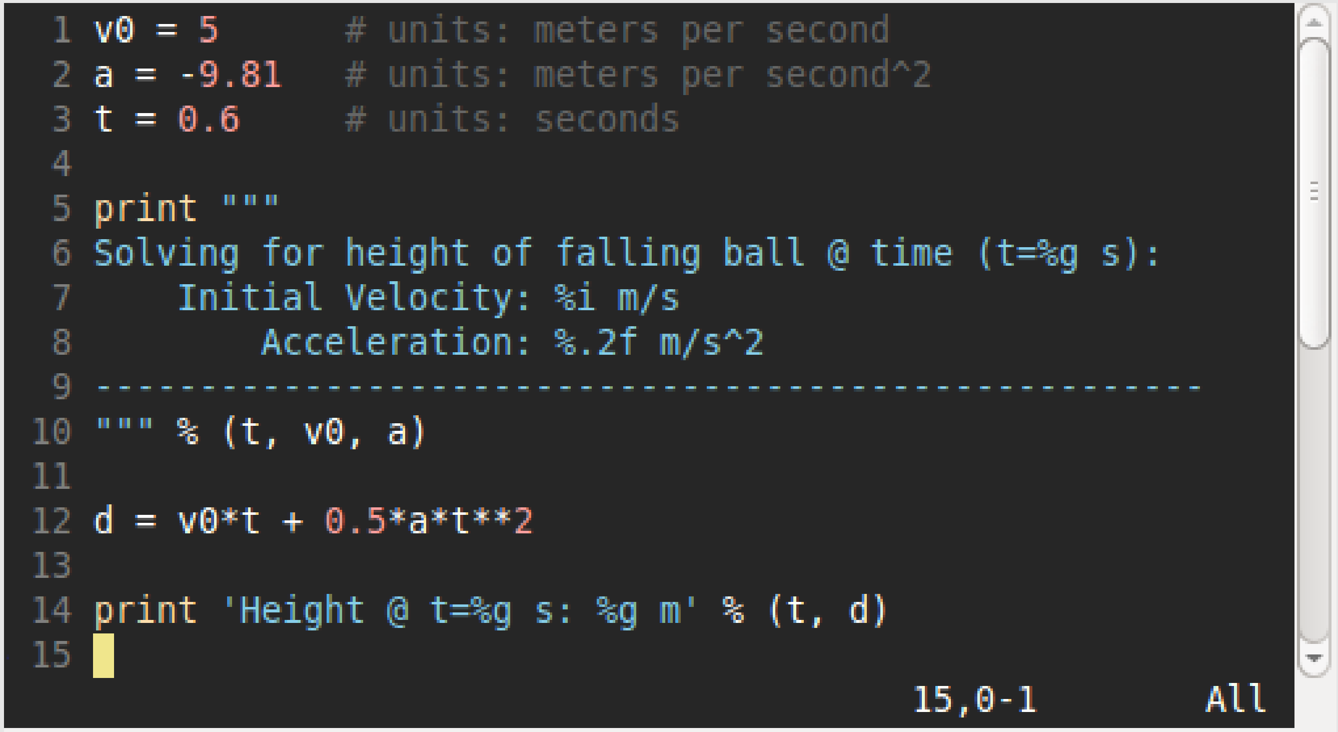
a – object's constant acceleration

Let's solve for the distance traveled by a upward moving ball at time t = 0.6 seconds, whose initial velocity at t = 0 was 5 m/s

Solving Numerical Problems

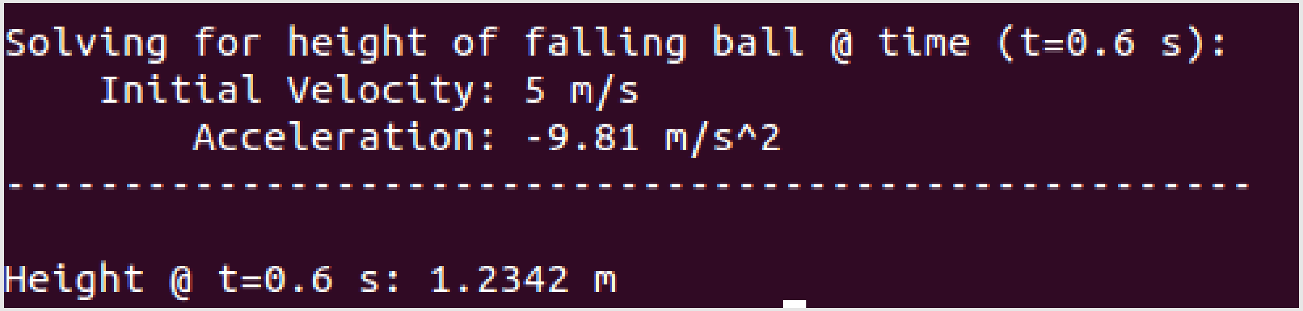
$$d(t)=v_0 t+\frac{1}{2}at^2$$

v_0	–	5 m/s
a	–	9.81 m/s^2
t	–	0.6 s
$d(t)$	–	???



```
v0 = 5      # units: meters per second
a = -9.81   # units: meters per second^2
t = 0.6     # units: seconds

print """
Solving for height of falling ball @ time (t=%g s):
    Initial Velocity: %i m/s
        Acceleration: %.2f m/s^2
-----------------------------------------------------
""" % (t, v0, a)

d = v0*t + 0.5*a*t**2

print 'Height @ t=%g s: %g m' % (t, d)
```

Output:

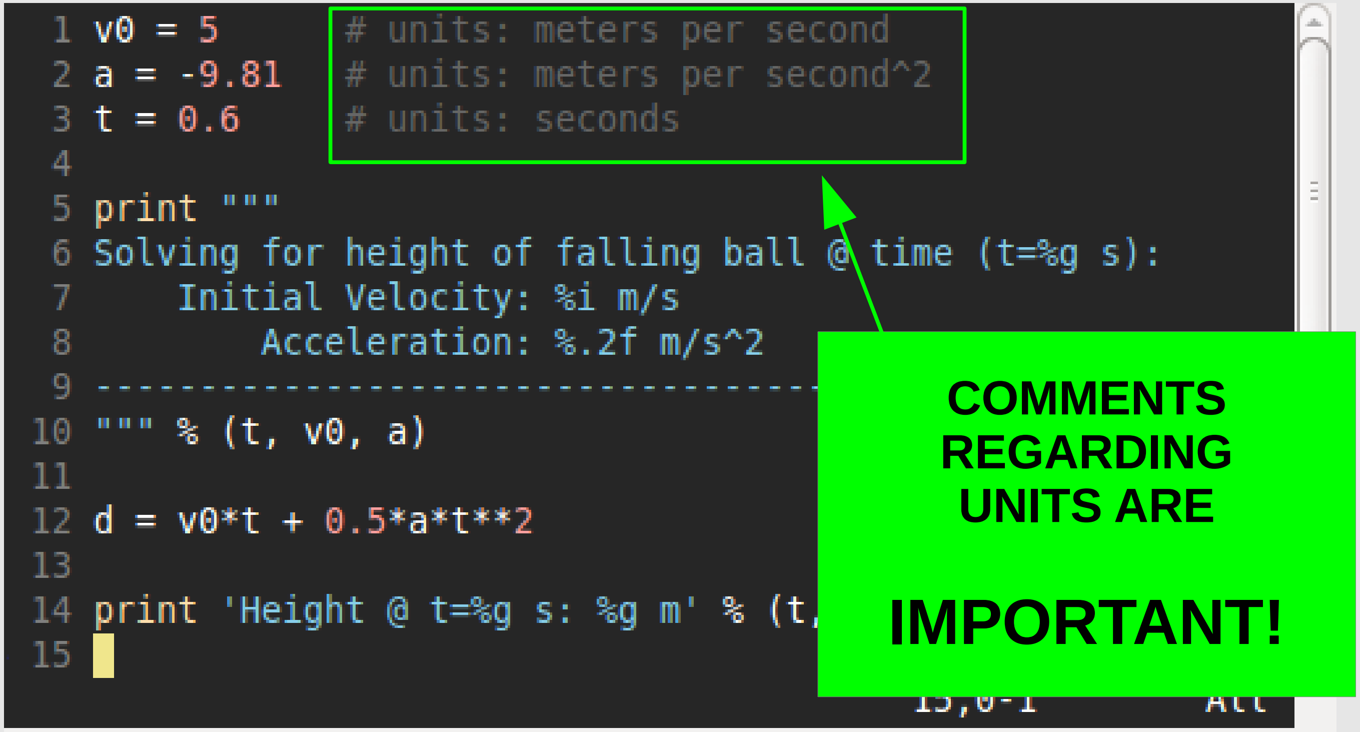


```
Solving for height of falling ball @ time (t=0.6 s):
    Initial Velocity: 5 m/s
        Acceleration: -9.81 m/s^2
-----------------------------------------------------

Height @ t=0.6 s: 1.2342 m
```

Solving Numerical Problems

$$d(t) = v_0 t + \frac{1}{2} a t^2$$

v_0	–	5 m/s
a	–	9.81 m/s^2
t	–	0.6 s
$d(t)$	–	???

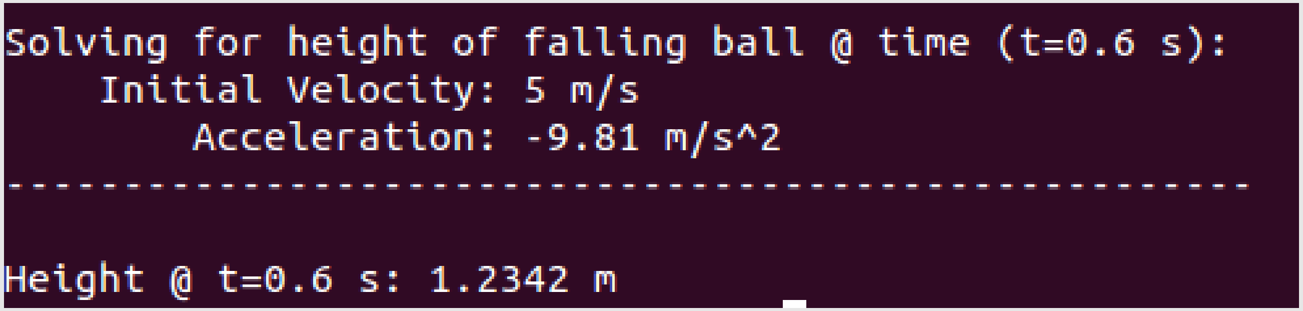


```
v0 = 5      # units: meters per second
a = -9.81   # units: meters per second^2
t = 0.6     # units: seconds

print """
Solving for height of falling ball @ time (t=%g s):
    Initial Velocity: %i m/s
        Acceleration: %.2f m/s^2
-----------------------------------
""" % (t, v0, a)

d = v0*t + 0.5*a*t**2

print 'Height @ t=%g s: %g m' % (t,

```

COMMENTS REGARDING UNITS ARE

IMPORTANT!

Output:

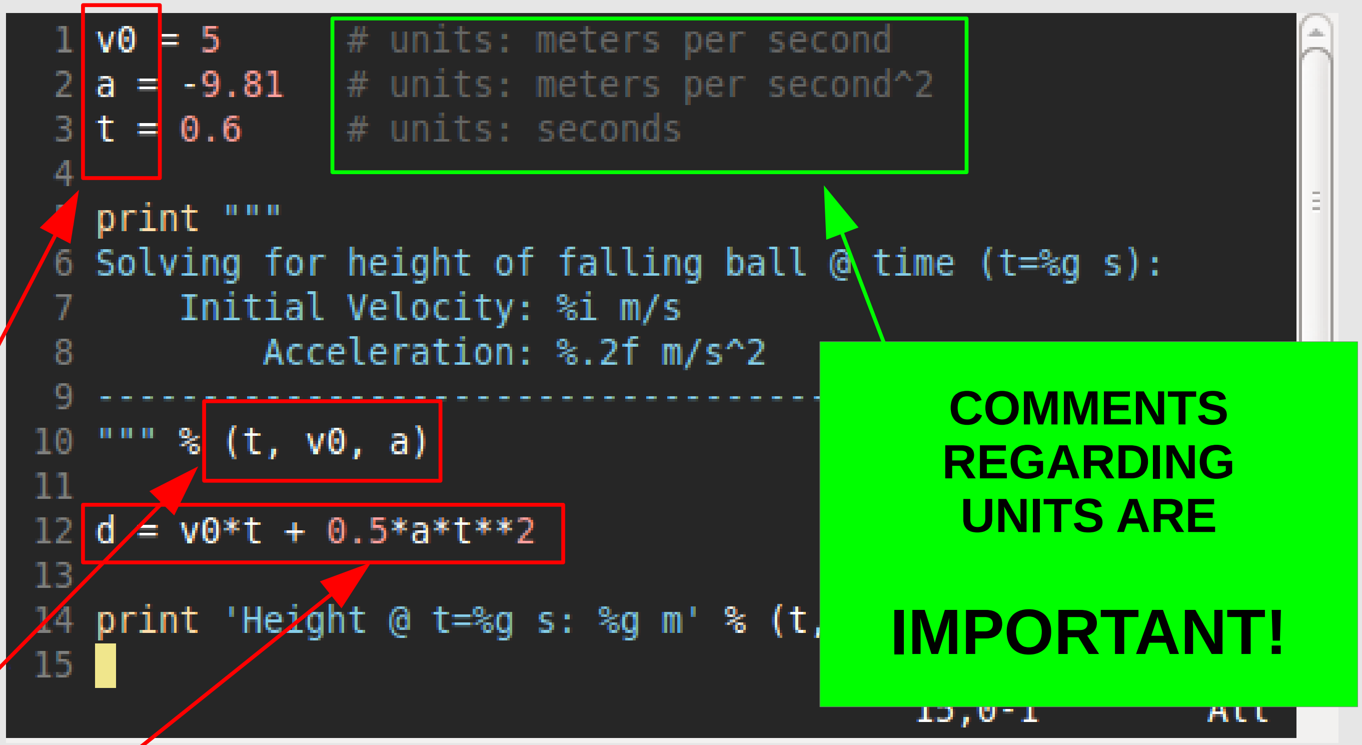


```
Solving for height of falling ball @ time (t=0.6 s):
    Initial Velocity: 5 m/s
        Acceleration: -9.81 m/s^2
-----------------------------------------------------

Height @ t=0.6 s: 1.2342 m
```

Solving Numerical Problems

$$d(t)=v_0 t+\frac{1}{2}at^2$$

v_0	–	5 m/s
a	–	9.81 m/s^2
t	–	0.6 s
$d(t)$	–	???



```
v0 = 5          # units: meters per second
a = -9.81       # units: meters per second^2
t = 0.6         # units: seconds

print """
Solving for height of falling ball @ time (t=%g s):
    Initial Velocity: %i m/s
        Acceleration: %.2f m/s^2
-----------------------------------------------------
""" % (t, v0, a)

d = v0*t + 0.5*a*t**2

print 'Height @ t=%g s: %g m' % (t,

```

COMMENTS REGARDING UNITS ARE

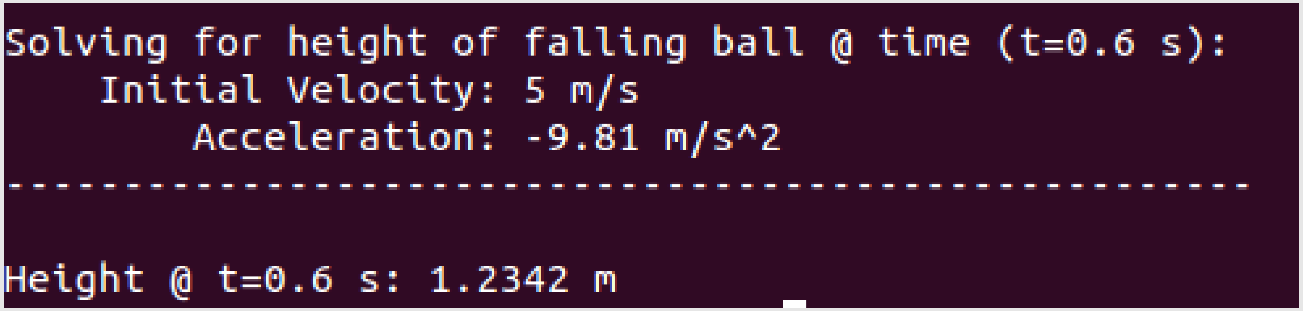
IMPORTANT!

UGH!

READS LIKE IT WAS WRITTEN BY A <u>MATHEMATICIAN</u>

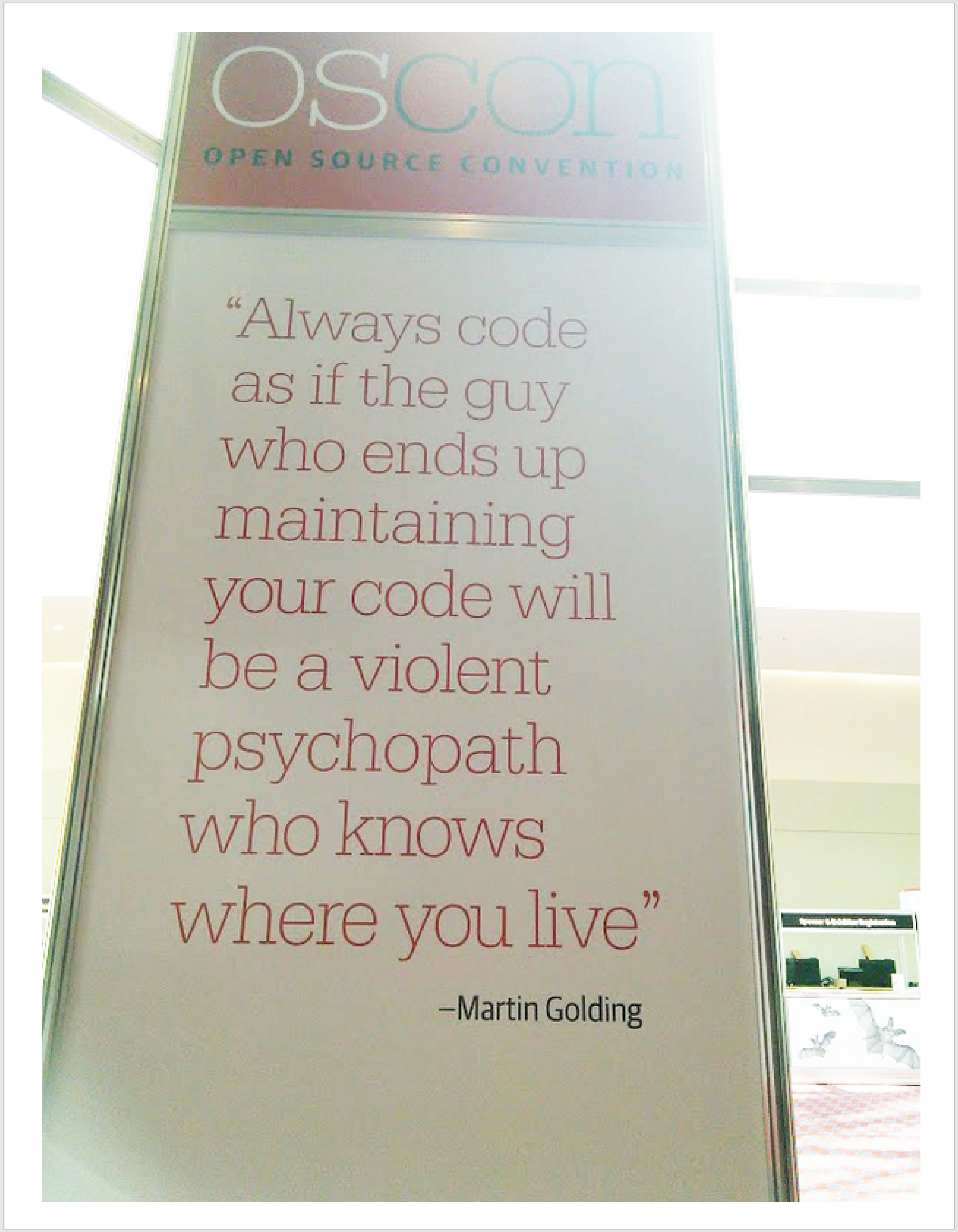
or somebody who doesn't speak English

Output:



```
Solving for height of falling ball @ time (t=0.6 s):
    Initial Velocity: 5 m/s
        Acceleration: -9.81 m/s^2
-----------------------------------------------------

Height @ t=0.6 s: 1.2342 m
```

Solving Numerical Problems



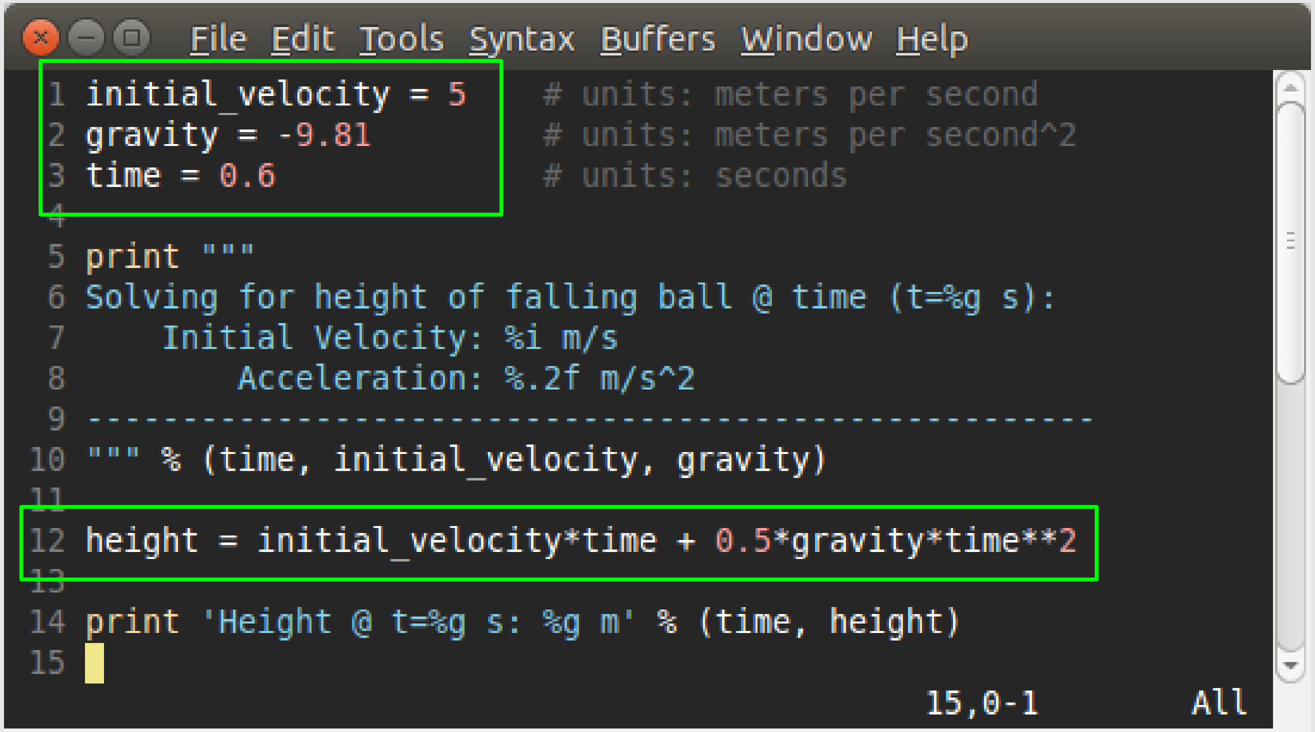
"Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live"

–Martin Golding

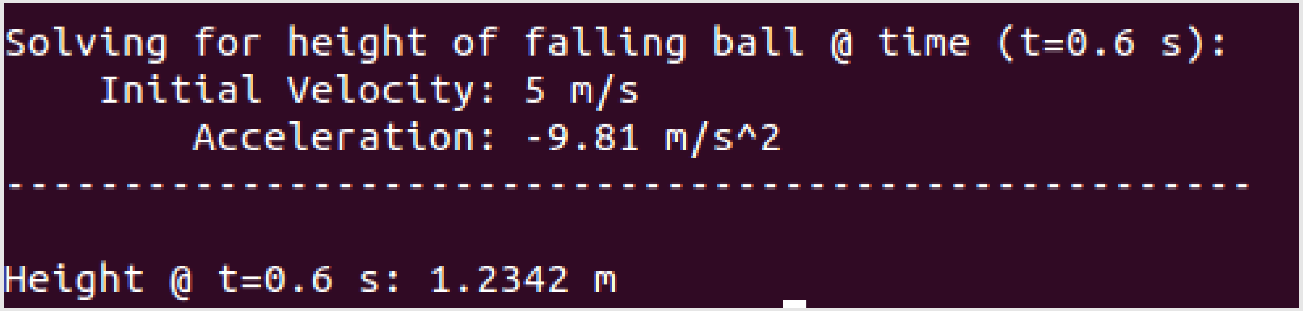
Solving Numerical Problems

$$d(t)=v_0 t+\frac{1}{2}at^2$$

v_0	–	5 m/s
a	–	9.81 m/s^2
t	–	0.6 s
$d(t)$	–	???



```
initial_velocity = 5    # units: meters per second
gravity = -9.81         # units: meters per second^2
time = 0.6              # units: seconds

print """
Solving for height of falling ball @ time (t=%g s):
    Initial Velocity: %i m/s
        Acceleration: %.2f m/s^2
-----------------------------------------------------
""" % (time, initial_velocity, gravity)

height = initial_velocity*time + 0.5*gravity*time**2

print 'Height @ t=%g s: %g m' % (time, height)
```



```
Solving for height of falling ball @ time (t=0.6 s):
    Initial Velocity: 5 m/s
        Acceleration: -9.81 m/s^2
-----------------------------------------------------

Height @ t=0.6 s: 1.2342 m
```

Solving Numerical Problems

It's very bad practice to
require your program to be edited

EVERY TIME

we want to try a different time *t*

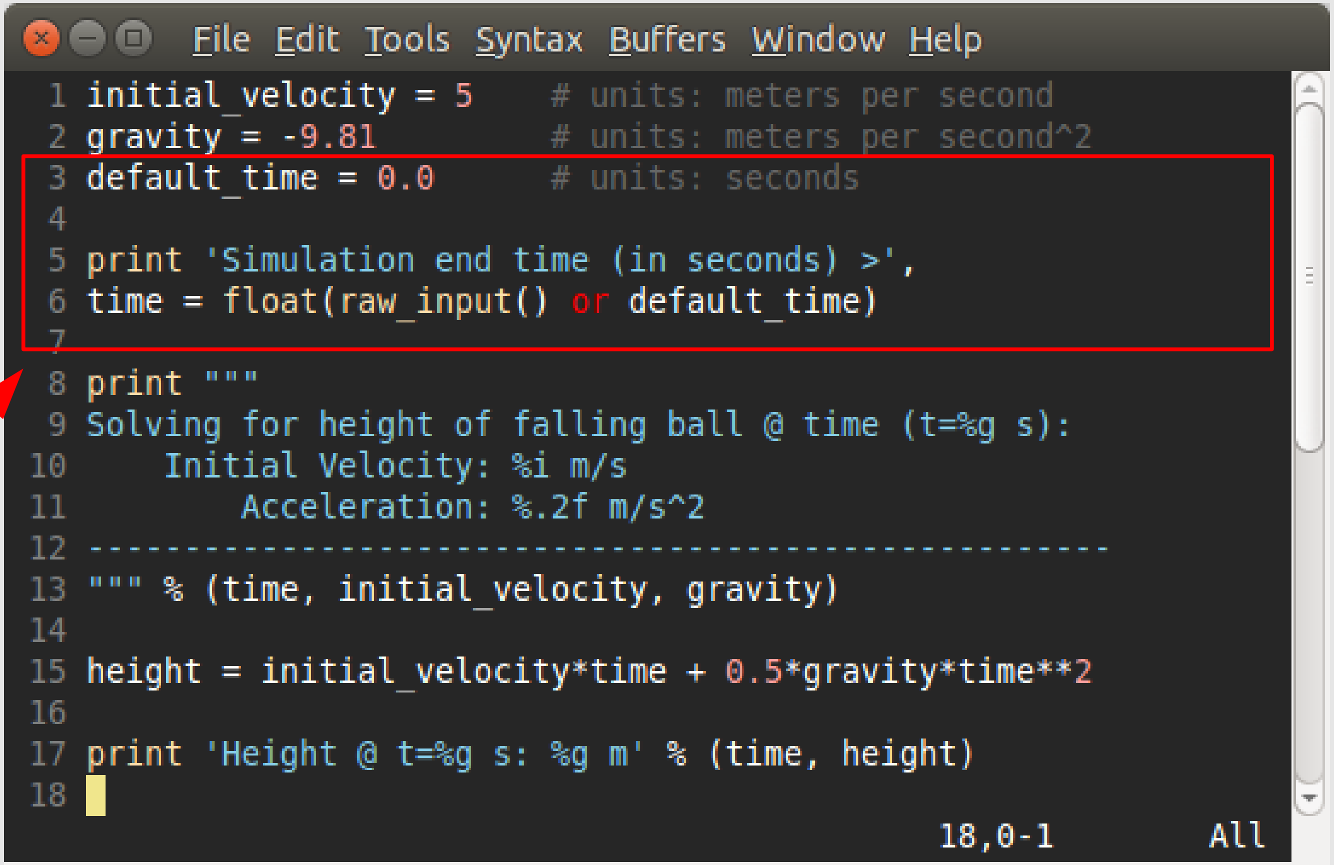
Let's spice this up by accepting user input!

Solving Numerical Problems

$$d(t)=v_0 t+\frac{1}{2}at^2$$

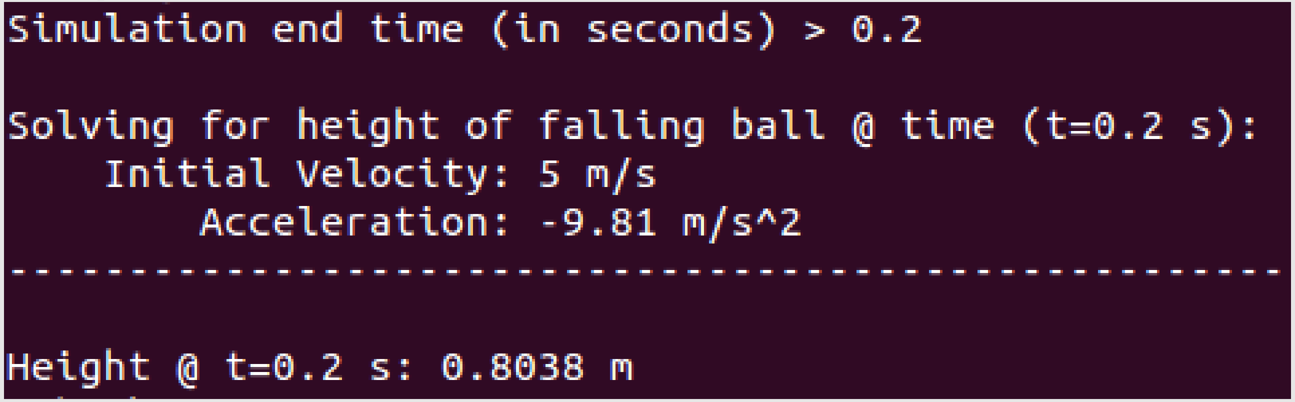
v_0	–	5 m/s
a	–	9.81 m/s^2
t	–	0.6 s
$d(t)$	–	???

How does this work?



```
initial_velocity = 5    # units: meters per second
gravity = -9.81         # units: meters per second^2
default_time = 0.0      # units: seconds

print 'Simulation end time (in seconds) >',
time = float(raw_input() or default_time)

print """
Solving for height of falling ball @ time (t=%g s):
    Initial Velocity: %i m/s
        Acceleration: %.2f m/s^2
-----------------------------------------------------
""" % (time, initial_velocity, gravity)

height = initial_velocity*time + 0.5*gravity*time**2

print 'Height @ t=%g s: %g m' % (time, height)
```

Output:

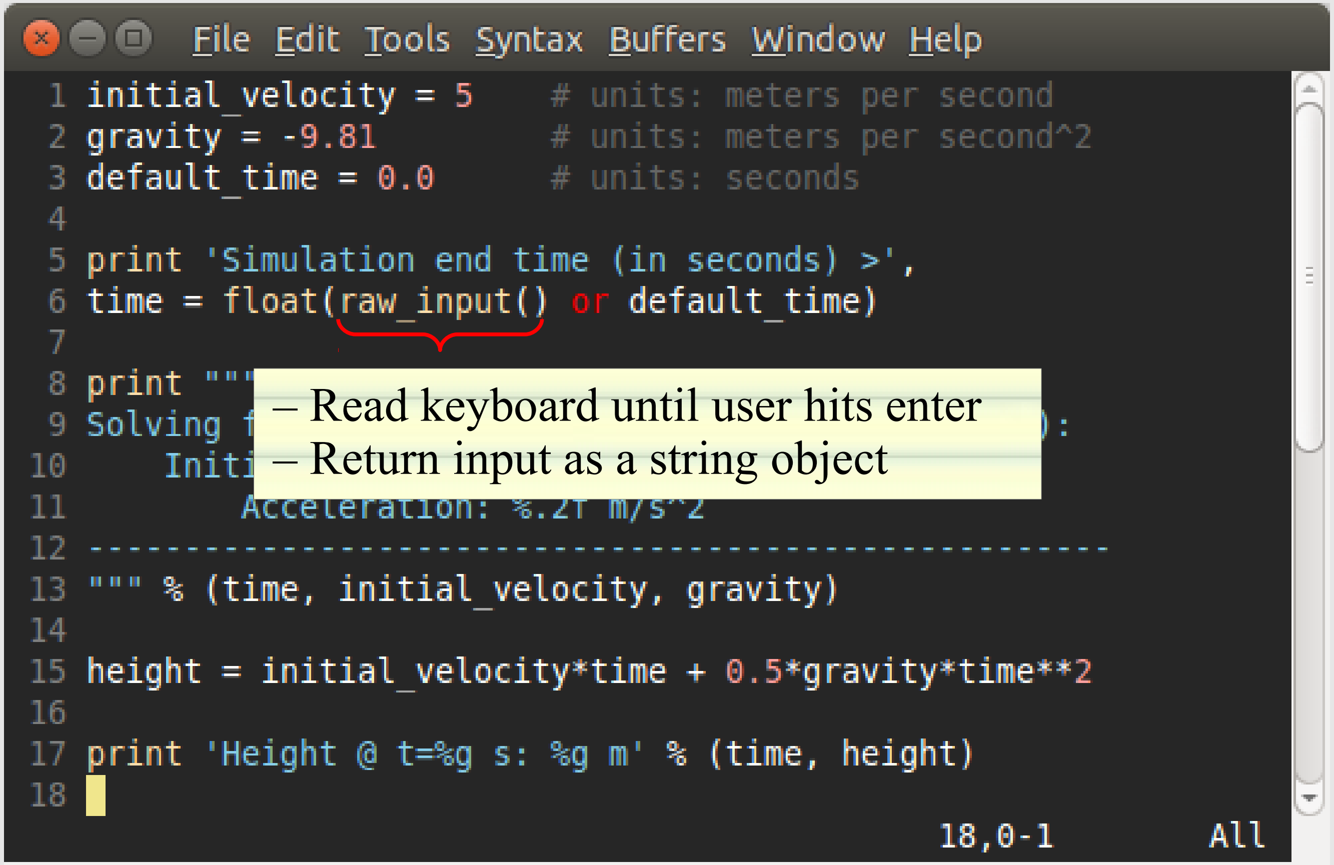


```
Simulation end time (in seconds) > 0.2

Solving for height of falling ball @ time (t=0.2 s):
    Initial Velocity: 5 m/s
        Acceleration: -9.81 m/s^2
-----------------------------------------------------


Height @ t=0.2 s: 0.8038 m
```

Solving Numerical Problems

$$d(t)=v_0 t+\frac{1}{2}at^2$$

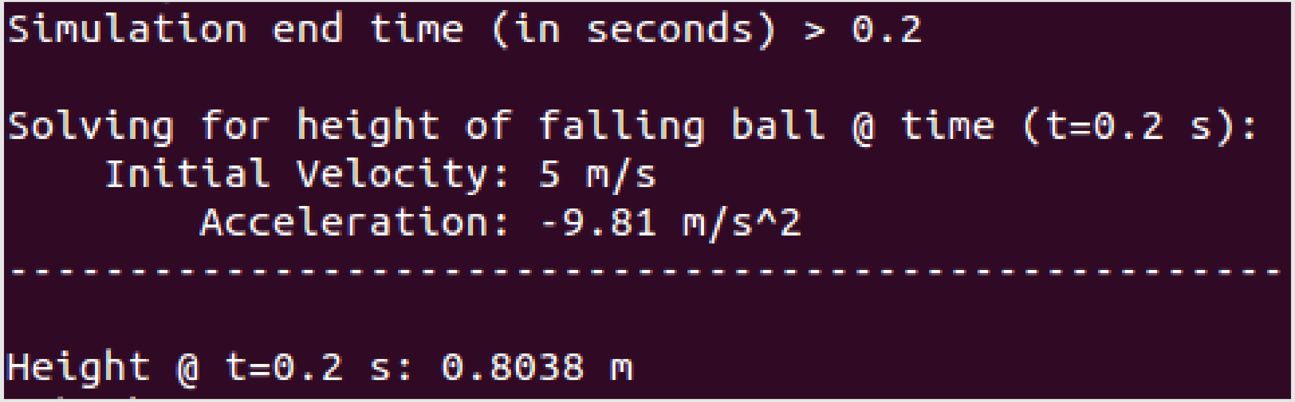
v_0	–	5 m/s
a	–	9.81 m/s^2
t	–	0.6 s
$d(t)$	–	???



```
initial_velocity = 5     # units: meters per second
gravity = -9.81          # units: meters per second^2
default_time = 0.0       # units: seconds

print 'Simulation end time (in seconds) >',
time = float(raw_input() or default_time)

print """
Solving f                                              ):
    Initi
        Acceleration: %.2f m/s^2
-----------------------------------------------------
""" % (time, initial_velocity, gravity)

height = initial_velocity*time + 0.5*gravity*time**2

print 'Height @ t=%g s: %g m' % (time, height)

```

– Read keyboard until user hits enter
– Return input as a string object

Output:

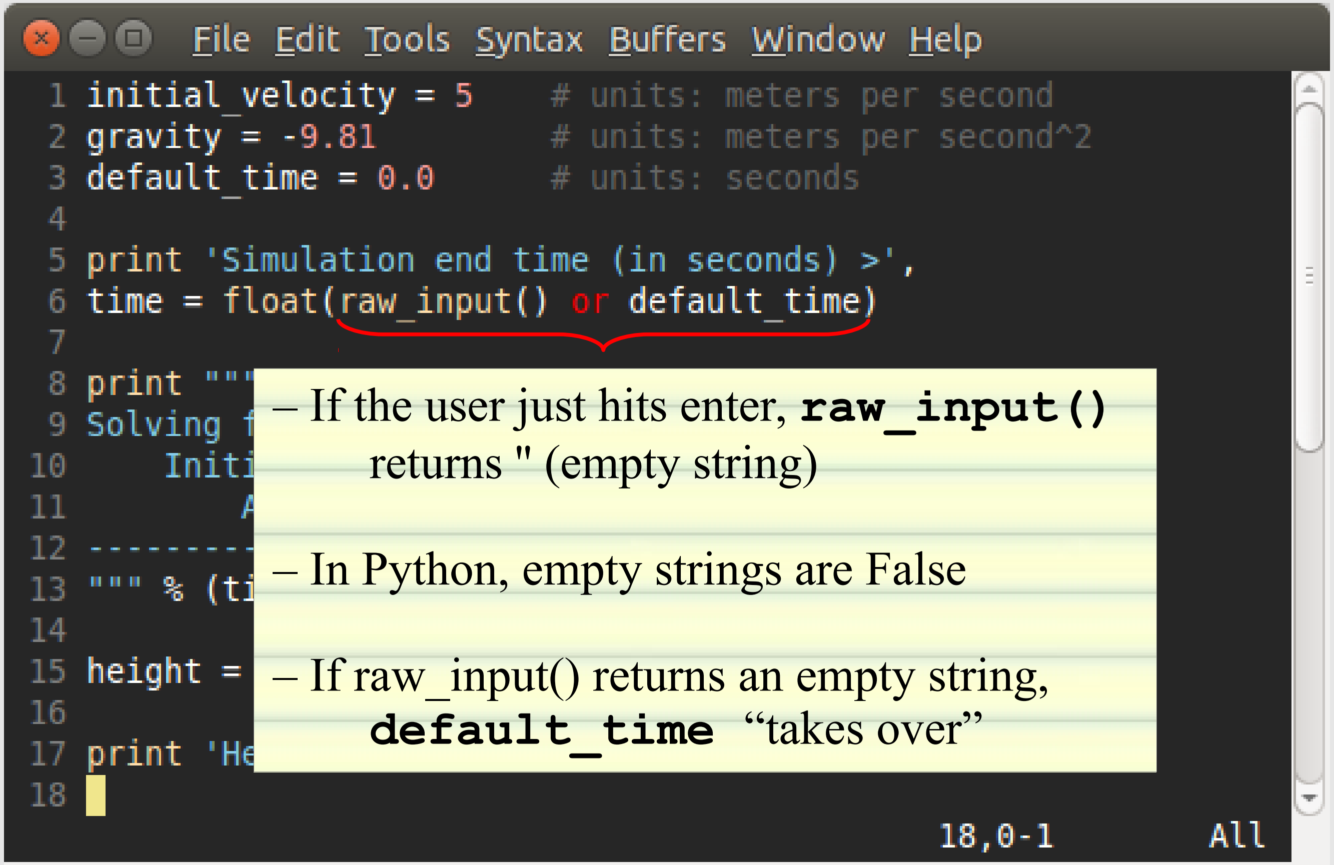


```
Simulation end time (in seconds) > 0.2

Solving for height of falling ball @ time (t=0.2 s):
    Initial Velocity: 5 m/s
        Acceleration: -9.81 m/s^2
-----------------------------------------------------


Height @ t=0.2 s: 0.8038 m
```

Solving Numerical Problems

$$d(t) = v_0 t + \frac{1}{2} a t^2$$

v_0	–	5 m/s
a	–	9.81 m/s²
t	–	0.6 s
$d(t)$	–	???



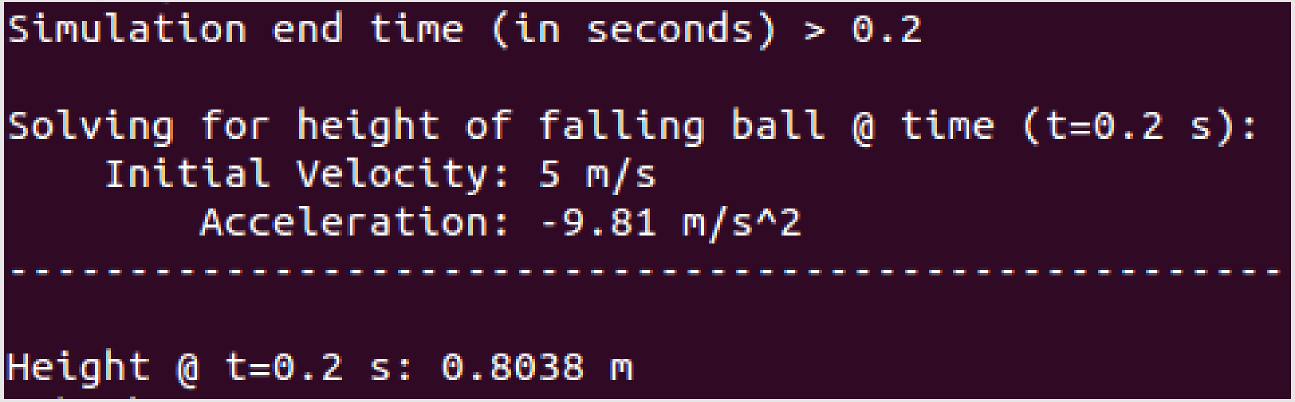
```
initial_velocity = 5     # units: meters per second
gravity = -9.81          # units: meters per second^2
default_time = 0.0       # units: seconds

print 'Simulation end time (in seconds) >',
time = float(raw_input() or default_time)

print """
Solving f
    Initi
        A
---------
""" % (ti

height =

print 'He

```

– If the user just hits enter, **`raw_input()`** returns '' (empty string)

– In Python, empty strings are False

– If raw_input() returns an empty string, **`default_time`** “takes over”

Output:

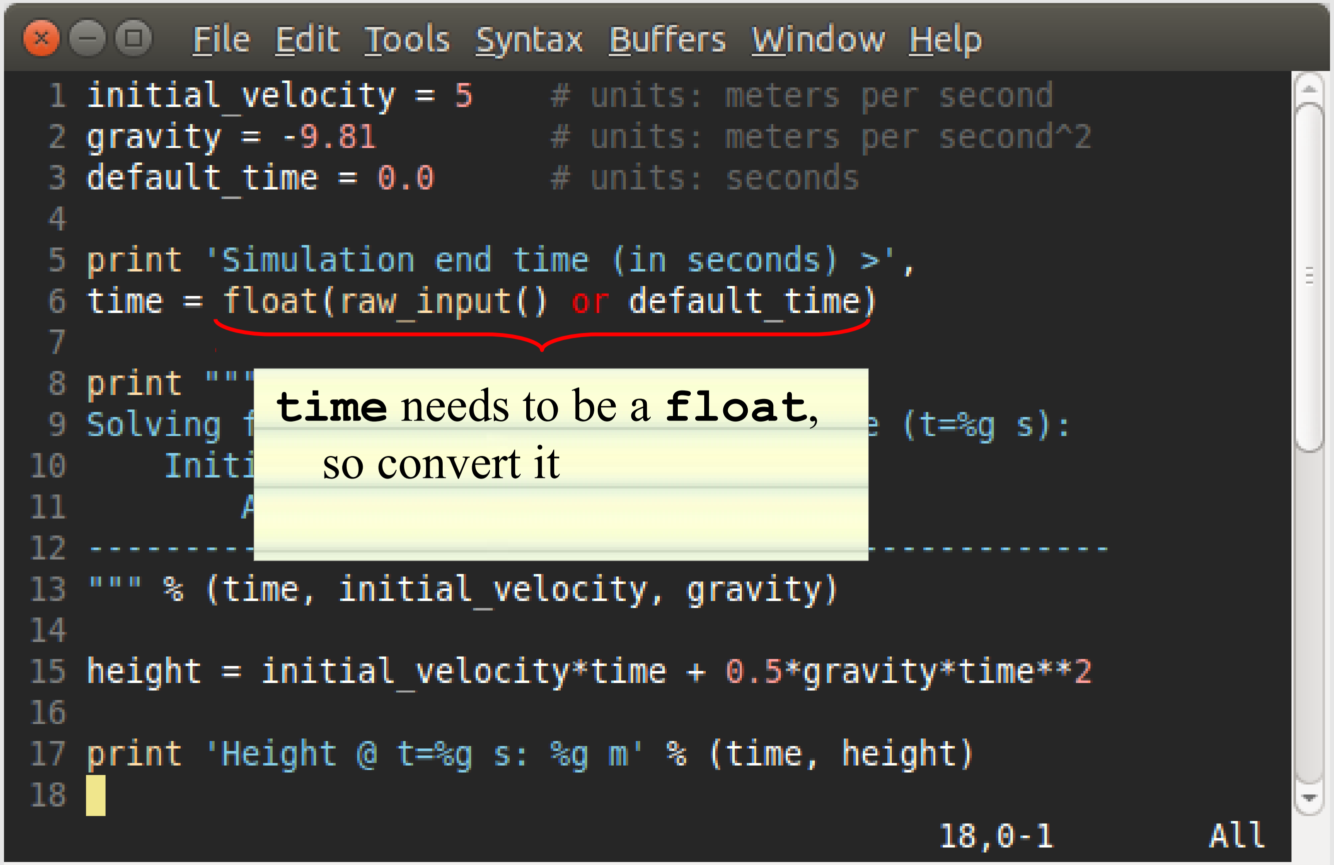


```
Simulation end time (in seconds) > 0.2

Solving for height of falling ball @ time (t=0.2 s):
    Initial Velocity: 5 m/s
        Acceleration: -9.81 m/s^2
-----------------------------------------------------

Height @ t=0.2 s: 0.8038 m
```

Solving Numerical Problems

$$d(t)=v_0 t+\frac{1}{2}at^2$$

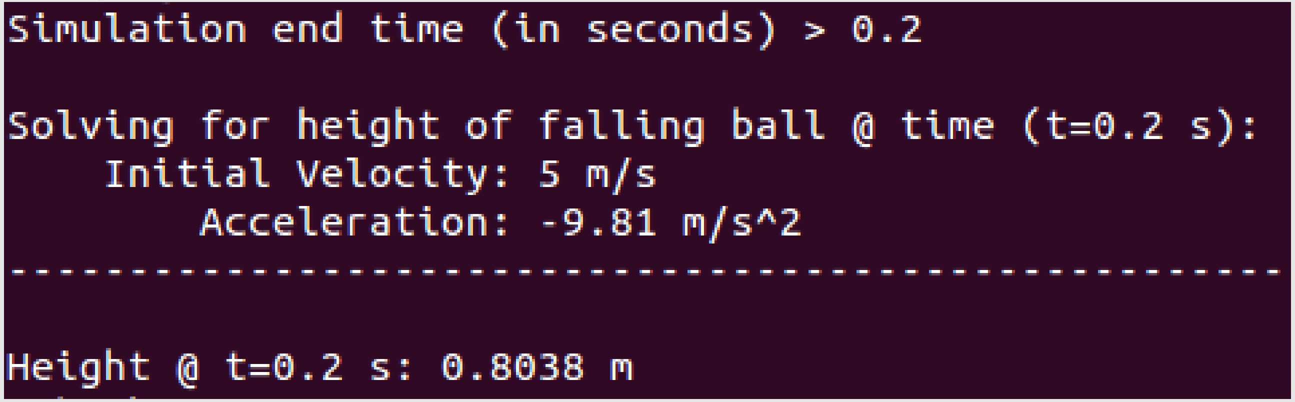
v_0	–	5 m/s
a	–	9.81 m/s^2
t	–	0.6 s
$d(t)$	–	???



```
initial_velocity = 5     # units: meters per second
gravity = -9.81          # units: meters per second^2
default_time = 0.0       # units: seconds

print 'Simulation end time (in seconds) >',
time = float(raw_input() or default_time)

print """
Solving f                                        e (t=%g s):
    Initi
        A
-------                                          -------------
""" % (time, initial_velocity, gravity)

height = initial_velocity*time + 0.5*gravity*time**2

print 'Height @ t=%g s: %g m' % (time, height)

```

time needs to be a **float**, so convert it

Output:



```
Simulation end time (in seconds) > 0.2

Solving for height of falling ball @ time (t=0.2 s):
    Initial Velocity: 5 m/s
        Acceleration: -9.81 m/s^2
-----------------------------------------------------


Height @ t=0.2 s: 0.8038 m
```

Solving Numerical Problems

Evaluating at single points in time is okay…

but we are ENGINEERS!

(we want a curve)

Sequence Attributes of Lists

Introduction to Lists

Lists are one of the three *Sequence Types* in Python

(the other two are tuples and strings)

If you "get" strings, lists are very similar

★ Lists are defined using square brackets [ ] – items are comma separated

```
>> my_list = [43, 23, 10, 5, 91]
>> print my_list
[43, 23, 10, 5, 91]
```

(you played with this a bit in lab)

Sequence Attributes of Lists

Introduction to Lists

Lists are one of the three *Sequence Types* in Python

(the other two are tuples and strings)

If you "get" strings, lists are very similar

★ Lists are defined using square brackets [ ] – items are comma separated

```
>> my_list = [43, 23, 10, 5, 91]
>> print my_list
[43, 23, 10, 5, 91]
```

(you played with this a bit in lab)

★ A single list can contain many different types of items

```
>> my_list = [84, “some words”, 1.234, 600, 'test']
>> print my_list
[84, 'some words', 1.234, 600, 'test']
```

Sequence Attributes of Lists

Introduction to Lists

Lists are one of the three *Sequence Types* in Python

(the other two are tuples and strings)

If you "get" strings, lists are very similar

★ All Sequence Types can be indexed – this includes Lists

```
>> my_list = [43, 23, 10, 5, 91]
>> first_item = my_list[0]
>> print first_item
43

>> print my_list[3]
5
```

(you played with this a bit in lab)

Sequence Attributes of Lists

Introduction to Lists

Lists are one of the three *Sequence Types* in Python

(the other two are tuples and strings)

If you "get" strings, lists are very similar

★ All Sequence Types can be indexed – this includes Lists

```
>> my_list = [43, 23, 10, 5, 91]
>> first_item = my_list[0]
>> print first_item
43

>> print my_list[3]
5
```

(you played with this a bit in lab)

★ Negative indices "wrap around" to the end and "go backwards"

```
>> my_list = [43, 23, 10, 5, 91]
>> last_item = my_list[-1]
>> print last_item
91

>> print my_list[-3]
10
```

List Methods

Introduction to Lists

Lists are one of the three *Sequence Types* in Python

(the other two are tuples and strings)

If you "get" strings, lists are very similar

★ **A new empty list can be defined easily**

```
>> my_list = []
```

List Methods

Introduction to Lists

Lists are one of the three *Sequence Types* in Python

(the other two are tuples and strings)

If you "get" strings, lists are very similar

★ **A new empty list can be defined easily**

```
>> my_list = []
```

★ **Adding new items to a list at runtime is also easy!**

```
>> my_list = []
>> my_list.append(40)
>> my_list.append('foo')
>> my_list.append(32.234)
>> print my_list
[40, 'foo', 32.234]

>> my_list.append('more words')
>> print my_list
[40, 'foo', 32.234, 'more words']
```

List Methods

Introduction to Lists

Lists are one of the three *Sequence Types* in Python

(the other two are tuples and strings)

If you "get" strings, lists are very similar

★ Removing arbitrary items from a list is simple

```
>> my_list = [98, 23, 'time of day', 32.99]
>> my_list.remove(23)
>> print my_list
[98, 'time of day', 32.99]

>> my_list.remove('time of day')
>> print my_list
[09, 32.99]
```

List Methods

Introduction to Lists

Lists are one of the three *Sequence Types* in Python

(the other two are tuples and strings)

If you "get" strings, lists are very similar

★ Removing arbitrary items from a list is simple

```
>> my_list = [98, 23, 'time of day', 32.99]
>> my_list.remove(23)
>> print my_list
[98, 'time of day', 32.99]

>> my_list.remove('time of day')
>> print my_list
[09, 32.99]
```

★ Lists are also sortable and reversible "in place"

```
>> my_list = [23, 40, 100, 21, 1, 59]
>> my_list.sort()
>> print my_list
[1, 21, 23, 40, 59, 100]

>> my_list.reverse()
>> print my_list
[100, 59, 40, 23, 21, 1]
```

Iterating Through Sequences

It's commonly desirable to
operate on all items in a list

one at a time

We need iterators and for-loops!

Brief Overview of Iterators

Introduction to Iterators

All *Sequence Types* in Python are iterables

★ **Iterables will return an iterator when passed to `iter()`**

```
>>> my_list = [98, 23, 'time of day', 32.99]
>>> my_iterator = iter(my_list)
```

Brief Overview of Iterators

Introduction to Iterators

All *Sequence Types* in Python are iterables

★ **Iterables will return an iterator when passed to `iter()`**

```
>>> my_list = [98, 23, 'time of day', 32.99]
>>> my_iterator = iter(my_list)
>>> my_iterator.next()
98
```

Brief Overview of Iterators

Introduction to Iterators

All *Sequence Types* in Python are iterables

★ **Iterables will return an iterator when passed to `iter()`**

```
>>> my_list = [98, 23, 'time of day', 32.99]
>>> my_iterator = iter(my_list)
>>> my_iterator.next()
98
>>> my_iterator.next()
23
```

Brief Overview of Iterators

Introduction to Iterators

All *Sequence Types* in Python are iterables

★ **Iterables will return an iterator when passed to `iter()`**

```
>>> my_list = [98, 23, 'time of day', 32.99]
>>> my_iterator = iter(my_list)
>>> my_iterator.next()
98
>>> my_iterator.next()
23
>>> my_iterator.next()
'time of day'
```

Brief Overview of Iterators

Introduction to Iterators

All *Sequence Types* in Python are iterables

★ **Iterables will return an iterator when passed to `iter()`**

```
>>> my_list = [98, 23, 'time of day', 32.99]
>>> my_iterator = iter(my_list)
>>> my_iterator.next()
98
>>> my_iterator.next()
23
>>> my_iterator.next()
'time of day'
>>> my_iterator.next()
32.99
```

Brief Overview of Iterators

Introduction to Iterators

All *Sequence Types* in Python are iterables

★ **Iterables will return an iterator when passed to `iter()`**

```
>>> my_list = [98, 23, 'time of day', 32.99]
>>> my_iterator = iter(my_list)
>>> my_iterator.next()
98
>>> my_iterator.next()
23
>>> my_iterator.next()
'time of day'
>>> my_iterator.next()
32.99
>>> my_iterator.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Brief Overview of Iterators

Introduction to Iterators

All *Sequence Types* in Python are iterables

★ Iterables will return an iterator when passed to `iter()`

```
>>> my_list = [98, 23, 'time of day', 32.99]
>>> my_iterator = iter(my_list)
>>> my_iterator.next()
98
>>> my_iterator.next()
23
>>> my_iterator.next()
'time of day'
>>> my_iterator.next()
32.99
>>> my_iterator.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

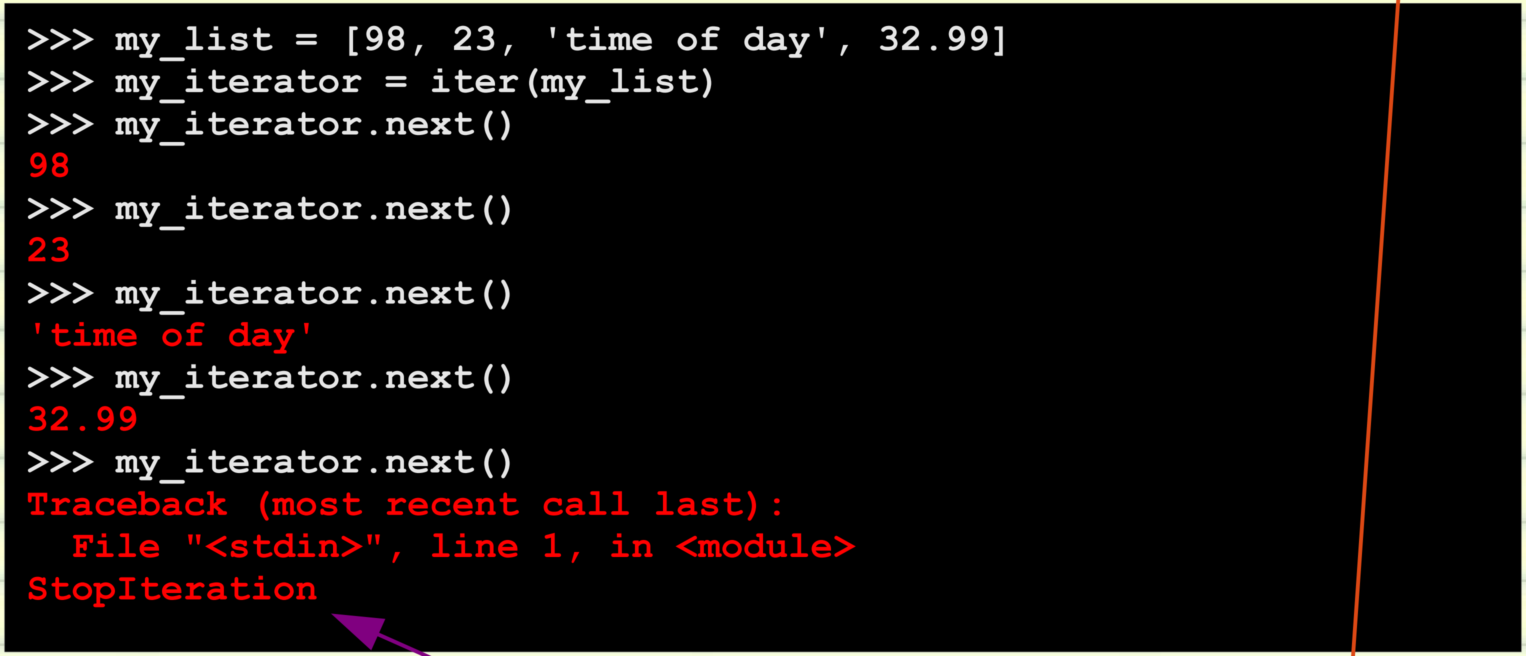
This is called an exception.
It means that something has happened
that could cause an error if not handled.
In this case, we exhausted the iterator.

Brief Overview of Iterators

Introduction to Iterators

All *Sequence Types* in Python are iterables

★ **Iterables will return an iterator when passed to `iter()`**



```
>>> my_list = [98, 23, 'time of day', 32.99]
>>> my_iterator = iter(my_list)
>>> my_iterator.next()
98
>>> my_iterator.next()
23
>>> my_iterator.next()
'time of day'
>>> my_iterator.next()
32.99
>>> my_iterator.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Exhausted iterators are dead for good.
If you need to iterate again, get another.

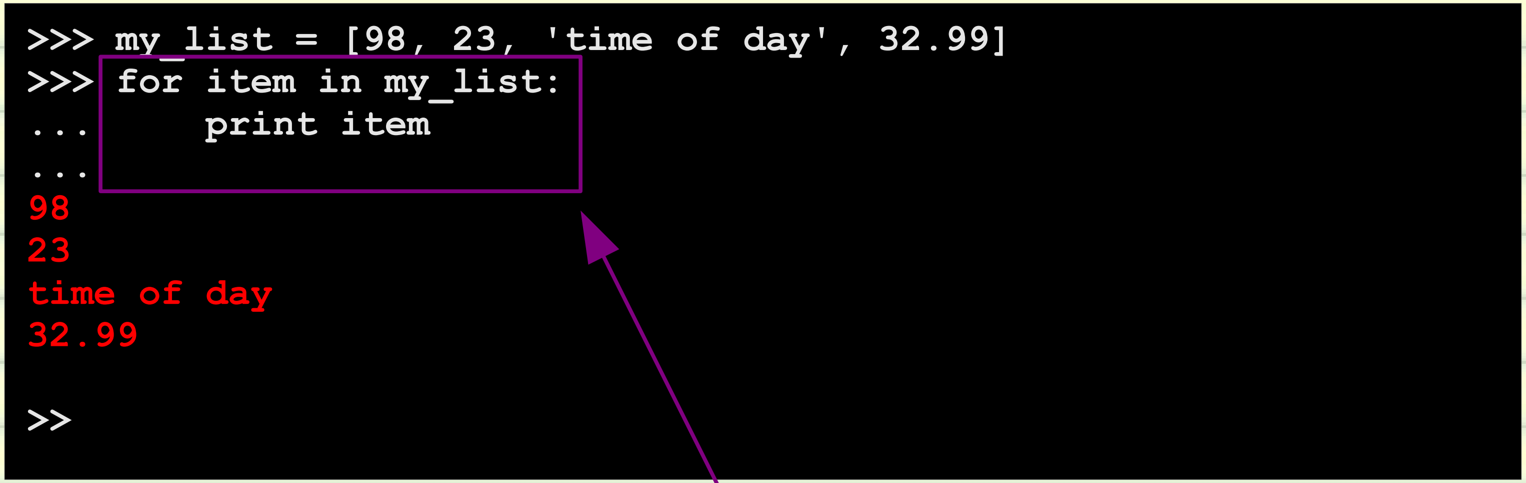
This is called an exception.
It means that something has happened
that could cause an error if not handled.
In this case, we exhausted the iterator.

The `for` loop

Introduction to for-loops

for-loops help us cycle through iterables and do work

★ **for-loops can be used to more effectively iterate over iterables**



```
>>> my_list = [98, 23, 'time of day', 32.99]
>>> for item in my_list:
...     print item
...
98
23
time of day
32.99

>>
```

The for-loop:

– “Secretly” calls `iter(my_list)` internally.

– “Secretly” does `item = iterator.next()`

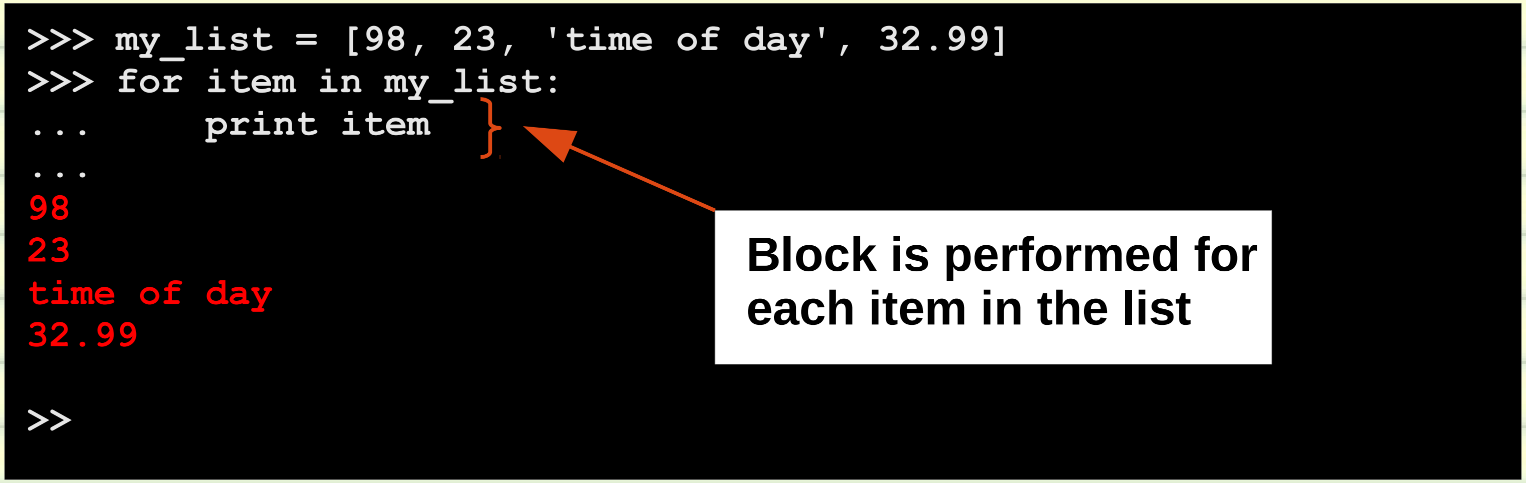
– terminates when it catches the `StopIteration` exception

The for loop

Introduction to for-loops

for-loops help us cycle through iterables and do work

★ **for-loops can be used to more effectively iterate over iterables**



```
>>> my_list = [98, 23, 'time of day', 32.99]
>>> for item in my_list:
...     print item
...
98
23
time of day
32.99

>>
```

Block is performed for each item in the list

The `for` loop

SOMETIMES…

we just want to loop a certain number of times

<u>For Example</u>
we would like to evaluate our physics problem multiple times for different values *t*

We need the `range()` function

Introducing the `range()` function

The (builtin-in) `range()` function

Simple. Generates a list containing a specified range of `ints`

★ **The simplest usage gives sequential `ints` starting from 0 & ending @ *N*-1**

```
>>> my_list = range(5)
>>> print my_list
[0, 1, 2, 3, 4]
```

Introducing the `range()` function

The (builtin-in) `range()` function

Simple. Generates a list containing a specified range of `ints`

★ **The simplest usage gives sequential `ints` starting from 0 & ending @ *N*-1**

```
>>> my_list = range(5)
>>> print my_list
[0, 1, 2, 3, 4]
```

★ **We can also specify a *starting point* other than zero**

```
>>> my_list = range(15, 20)
>>> print my_list
[15, 16, 17, 18, 19]
```

Introducing the `range()` function

The (builtin-in) `range()` function

Simple. Generates a list containing a specified range of `ints`

★ **The simplest usage gives sequential `ints` starting from 0 & ending @ *N*-1**

```
>>> my_list = range(5)
>>> print my_list
[0, 1, 2, 3, 4]
```

★ **We can also specify a *starting point* other than zero**

```
>>> my_list = range(15, 20)
>>> print my_list
[15, 16, 17, 18, 19]
```

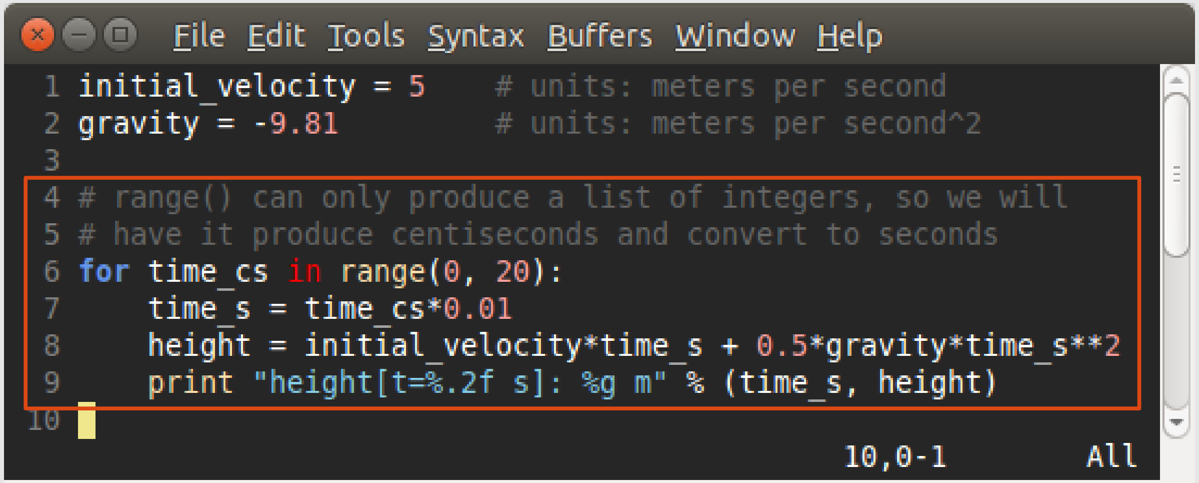
★ **We can also specify a *step***

```
>>> my_list = range(15, 30, 3)
>>> print my_list
[15, 18, 21, 24, 27]
```

Brief Overview of Iterators

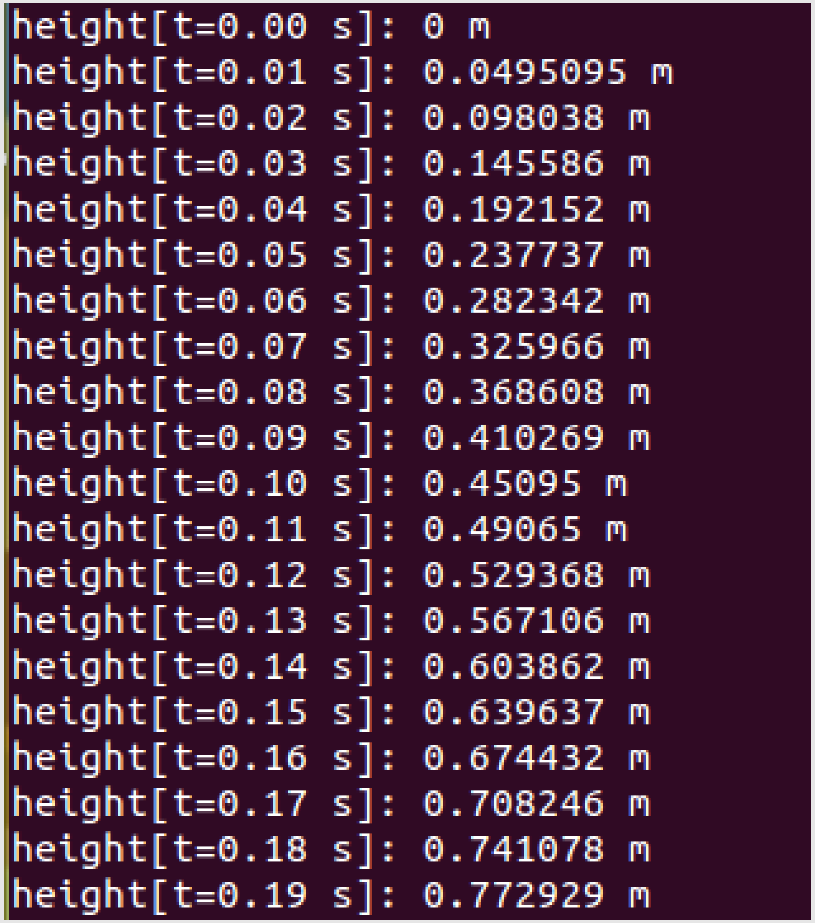
Let's update our program

Brief Overview of Iterators



```python
initial_velocity = 5    # units: meters per second
gravity = -9.81         # units: meters per second^2

# range() can only produce a list of integers, so we will
# have it produce centiseconds and convert to seconds
for time_cs in range(0, 20):
    time_s = time_cs*0.01
    height = initial_velocity*time_s + 0.5*gravity*time_s**2
    print "height[t=%.2f s]: %g m" % (time_s, height)
```

Output:



```
height[t=0.00 s]: 0 m
height[t=0.01 s]: 0.0495095 m
height[t=0.02 s]: 0.098038 m
height[t=0.03 s]: 0.145586 m
height[t=0.04 s]: 0.192152 m
height[t=0.05 s]: 0.237737 m
height[t=0.06 s]: 0.282342 m
height[t=0.07 s]: 0.325966 m
height[t=0.08 s]: 0.368608 m
height[t=0.09 s]: 0.410269 m
height[t=0.10 s]: 0.45095 m
height[t=0.11 s]: 0.49065 m
height[t=0.12 s]: 0.529368 m
height[t=0.13 s]: 0.567106 m
height[t=0.14 s]: 0.603862 m
height[t=0.15 s]: 0.639637 m
height[t=0.16 s]: 0.674432 m
height[t=0.17 s]: 0.708246 m
height[t=0.18 s]: 0.741078 m
height[t=0.19 s]: 0.772929 m
```

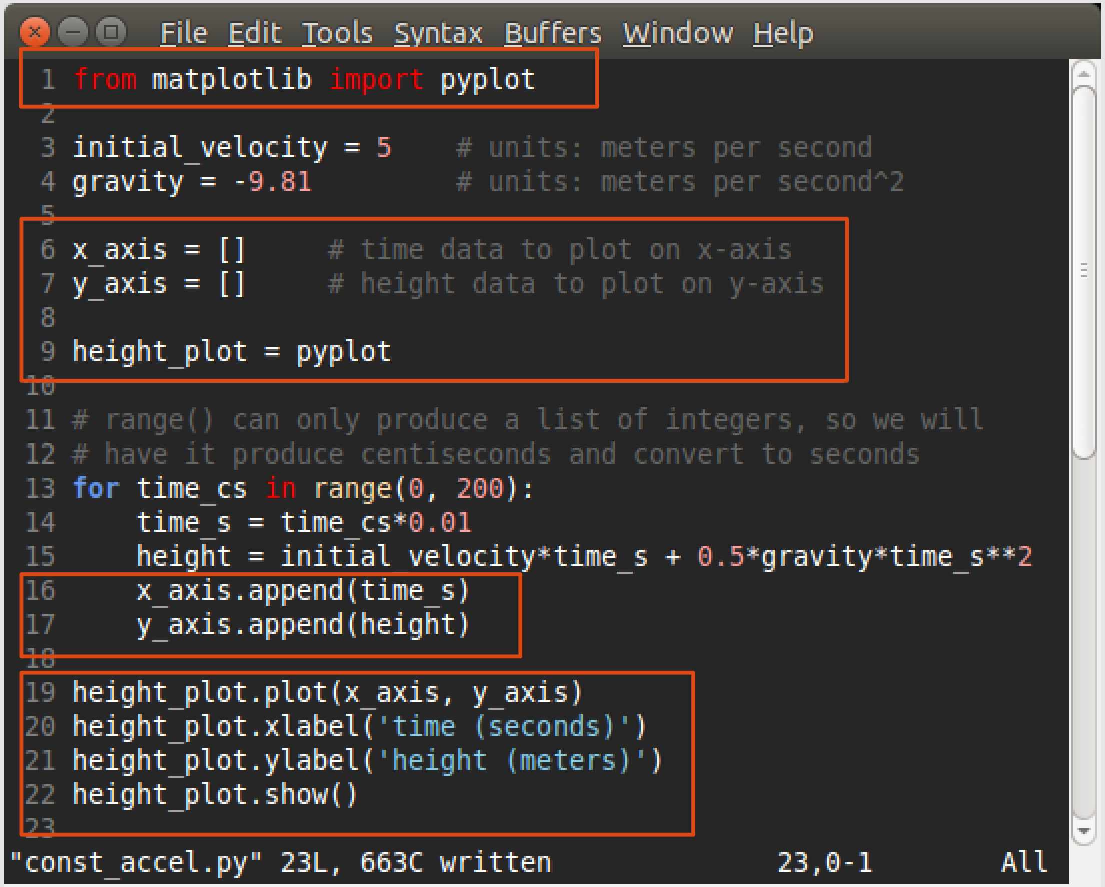
Brief Overview of Iterators

Now we want to plot

Let's `import` a module that extends Python

with plotting capabilities

Solving Numerical Problems



```
from matplotlib import pyplot

initial_velocity = 5     # units: meters per second
gravity = -9.81          # units: meters per second^2

x_axis = []      # time data to plot on x-axis
y_axis = []      # height data to plot on y-axis

height_plot = pyplot

# range() can only produce a list of integers, so we will
# have it produce centiseconds and convert to seconds
for time_cs in range(0, 200):
    time_s = time_cs*0.01
    height = initial_velocity*time_s + 0.5*gravity*time_s**2
    x_axis.append(time_s)
    y_axis.append(height)

height_plot.plot(x_axis, y_axis)
height_plot.xlabel('time (seconds)')
height_plot.ylabel('height (meters)')
height_plot.show()

"const_accel.py" 23L, 663C written                  23,0-1        All
```

Output:

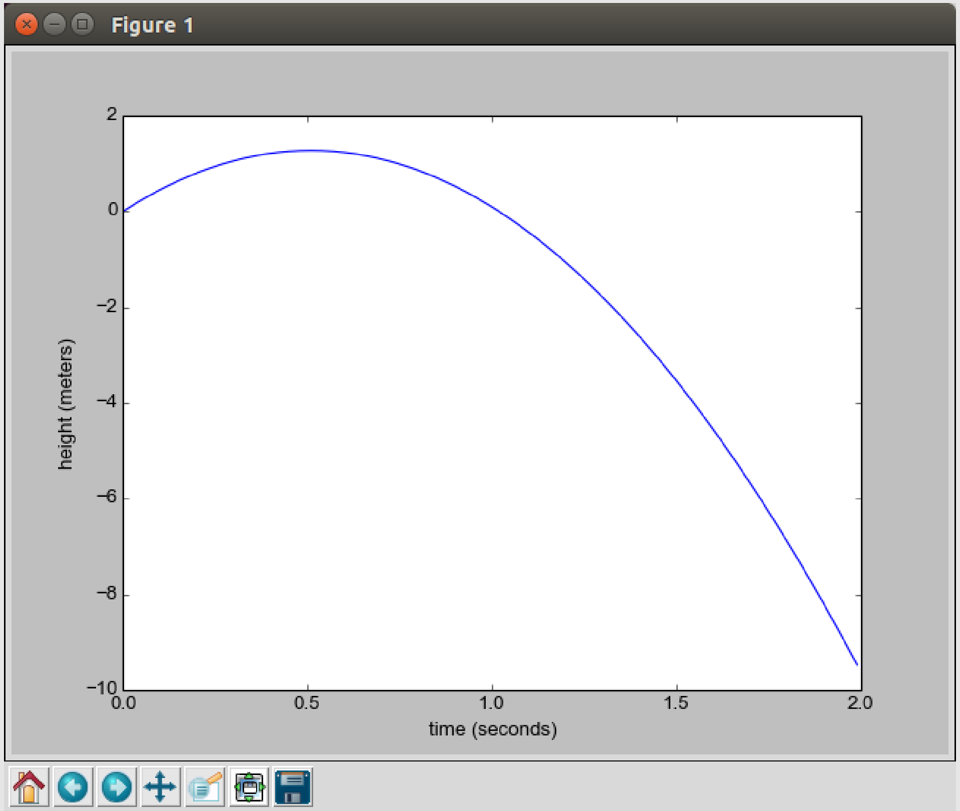


Figure 1
height (meters)
time (seconds)