Learning to Program with Python

Here we will use the python programming language to make a game of hangman, starting from scratch, working on a Macintosh. Python comes with OS X, so nothing special needs to be installed to follow along on your Mac. To use python on Windows, you can download and install python here.

It takes a couple hours to learn enough programming to make a simple game.

We will learn about:

- Memory and naming
- Computer arithmetic
- Using and learning libraries
- How to make a program
- Input and output
- Loops and choices
- Connecting to the internet

At the end we will have a game we can play.

This page was originally posted at http://davidbau.com/python for teaching a small group of third-graders.

Running Python

Go to the "Utilities" folder on your Mac and run "Terminal." It will look something like this:

```
Last login: Mon Jan 15 19:25:31 on ttyp1
Welcome to Darwin!
laptop:~ student$ 
```

Now type "python" and press the return button. Your screen should look like this now:

```
laptop:~ student$ python
Python 2.3.5 (#1, Jan 13 2006, 20:13:11)
[GCC 4.0.1 (Apple Computer, Inc. build 5250)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

The >>> means that python is waiting for you to program something.

Keeping a Secret

Python can remember things, so let's tell it to remember our secret word.

```
>>> secret = 'crocodile'
>>> 
```

You can tell it to print something by saying print.

```
>>> print secret
crocodile
>>> 
```

Python will remember the secret until you quit. So you can always go back and print your secret again by saying "print secret."

Now try to print something python doesn't know.

```
>>> print number
Traceback (most recent call last):
  File "", line 1, in
NameError: name 'number' is not defined
>>> 
```

Don't worry. This is fine. You can just teach python what "number" is and try again.

```
>>> number = 43
>>> print number
43
>>> 
```

Computers are Good Calculators

Your computer is better than any calculator at doing math. Let's try some.

```
>>> print 2+33+66
101
>>> print 33333333 * 44444444
1481481451851852
>>> n=123456789
>>> print n*n*n
1881676371789154860897069
>>> 
```

In Python, plus and minus are normal but times and divide are done using the * and / symbol. Some other symbols to know:

+	plus
-	minus
*	times
/	divide, rounding down

x = 95	save 95 as x
x == 24	is x equal to 24?
x < 24	is x less than 24?
x > 24	is x more than 24?

len(word)	the length of word
str(num)	turns num into a string of digits
int(digits)	makes a number from a string

What will it do when we say "len(str(1234))"?

Try your own fancy formulas. Don't worry if you get errors.

Picking a Random Number

Python comes with libraries of functions that do almost anything.

Let's tell Python to pick a random number. Get the "random" library by typing this:

```
>>> import random
>>> 
```

Now we can say "random.randint(1,100)" to pick a random number from 1 to 100.

```
>>> random.randint(1,100)
95
>>> 
```

Which number did you get?

You can learn about libraries by saying help. Try typing this:

```
>>> help(random)
Help on module random:

NAME
    random - Random variable generators.
(...and on...)
```

It will show lots of information, and you can press the space bar to see a screenful at a time. Eventually you will see a long list of all the functions in the library.

We don't have time to read it all now, so you can press "q" when you have seen enough.

You can use help() to ask about most things in python.

Words and Numbers

What do you think happens when you try to do math with words?

```
>>> print 'red' + 'yellow'
redyellow
>>> print 'red' * 3
redredred
>>> print 'red' + 3
Traceback (most recent call last):
  File "", line 1, in
TypeError: cannot concatenate 'str' and 'int' objects
>>> 
```

If you add two words, it sticks them together. If you times a word by 3, it makes three copies.

Python doesn't know how to add a word and a number, so it says "cannot concatenate 'str' and 'int' objects." A word that you put in quotes is just a string of letters called a "str" in python. Numbers that don't have a decimal point are integers and are called "int" in python.

You can't add a str and an int. But you can turn a number into a string if you use the str() function.

```
>>> print 'red' + str(3)
red3
>>> 
```

Using "for" to Repeat

You can repeat something with the "for" command.

If you say "for *something* in *something*:" with a colon (:) at the end, python will look for any indented lines afterwards and repeat them. Try this:

```
>>> for letter in secret:
...   print letter
...
c
r
o
c
o
d
i
l
e
>>> ▯
```

Think about your program - it is saying "for every letter in the secret, do this next thing." The computer repeats "print letter" nine times, once for each letter.

If it didn't work for you, don't worry. Press return and try again. **Check that you put in the extra spaces and the punctuation.** Spaces at the beginning of a line **do** make a difference in python.

Of course, in our game we shouldn't give away our secret word right away. So once you have the hang of it, try this:

```
>>> for letter in secret:
...   print '_',
_ _ _ _ _ _ _ _ _
>>> ▯
```

The comma at the end of print tells it not to start a new line.

Making Your Own Program

A program is just a file with code in it. Let's make one.

1. Start TextEdit from your Applications folder.
2. Go to the "Format" menu and say "Make Plain Text".
3. Now type this in:

```
#!/usr/bin/env python

print 'time to play hangman'
```

The first line tells the computer which what language the program is written in: python.

After that, you can write as many lines of python code as you like.

Save the file with the name "game.py" on your desktop.

Running Your Program

Go back to Terminal. If you are still in python, press control-D to get out. Arrange your windows so you can see TextEdit and Terminal at the same time.

The magic way to make a file into a program is "chmod +x *filename*" like this:

```
laptop:~ student$ chmod +x Desktop/game.py
laptop:~ student$ ▯
```

The "chmod" command stands for "change mode" and "+x" means "make it e**x**ecutable." You just marked your file as a program.

Now you can run your program by typing its name.

```
laptop:~ student$ Desktop/game.py
time to play hangman
laptop:~ student$ ▯
```

That's it. You've made your first program!

Programs are just Scripts

A program is just a script that a computer reads as quickly as it can. If we put a lot of lines of code into a program, the computer will do them all, one after the other.

Try typing these extra lines into your TextEdit. Don't forget to save the file again.

```
#!/usr/bin/env python

print 'time to play hangman'
secret = 'crocodile'
for letter in secret:
  print '_',
```

Now run the program in Terminal.

You can use the up-arrow if you don't want to bother typing the name of your program again. After you have the name of your program, press return.

```
laptop:~ student$ Desktop/game.py
time to play hangman
_ _ _ _ _ _ _ _ _
laptop:~ student$ █
```

See what happens? Everthing in your program ran very quickly.

Using "if" to Choose

In our hangman game, we should show where any guessed letters are. To decide whether to print a line or a letter, we will need to use "if" and "else".

Try changing your program inside TextEdit like this:

```
#!/usr/bin/env python

print 'time to play hangman'
secret = 'crocodile'
guesses = 'aeiou'

for letter in secret:
  if letter in guesses:
    print letter,
  else:
    print '_',
```

Don't forget to line everything up, and remember to save it.

What happens when you run it?

The line "**if** *something* in *something*:" makes a choice. If the letter is in our guesses, it prints the letter. Otherwise it prints a little line ("**else:**" is how you say "otherwise" in python).

Since the whole thing is under the "**for** *something* in *something*", this choice is repeated for every letter.

```
laptop:~ student$ Desktop/game.py
_ _ o _ o _ i _ e
laptop:~ student$ 
```

Check your spelling and spacing and punctuation if you get errors. Take your time to get it to work.

Input with "raw_input"

Our game is no good if you can't guess. To let the player guess we will use a function called "raw_input()."

It works like this:

```
answer = raw_input('my question')
```

So try changing your program to look like this:

```
#!/usr/bin/env python

print 'time to play hangman'
secret = 'crocodile'
guesses = 'aeiou'

guess = raw_input('guess a letter: ')
guesses += guess

for letter in secret:
  if letter in guesses:
    print letter,
  else:
    print '_',
```

The "guesses += guess" line add the guess to the string of guesses. If the string of guesses was "aeiou" and the new guess is "c", then the string of guesses will become "aeiouc".

Let's run it.

```
laptop:~ student$ Desktop/game.py
guess a letter: c
c _ o _ o _ i _ e
laptop:~ student$ 
```

You can guess any letter and it will show it to you.

Using "while" to Repeat

We need to let the player take more than one turn.

The "while" command can repeat our program until the player is out of turns.

```
#!/usr/bin/env python

print 'time to play hangman'
secret = 'crocodile'
guesses = 'aeiou'
turns = 5

while turns > 0:
  for letter in secret:
    if letter in guesses:
      print letter,
    else:
      print '_',

  print
  guess = raw_input('guess a letter: ')
  guesses += guess
  turns -= 1
```

You need to indent everything under the "while" command to make this work. So you will need to add some spaces in front of most of your program.

Let's also move the guessing after the hint instead of before.

The command "turns -= 1" means subtract one from "turns," so if it used to be 5, it will be 4. Then the next time around it will be 3 and so on. When turns is finally zero, the "while" command will stop repeating.

Try running your program. Does it work?

Improving Our Game

We can already play our game. Now we need to fix it up so that it is fun.

1. The player should win right away when there are no missing letters.
2. The player should only lose a turn on a wrong guess.
3. When the player loses, the game should tell the secret.

Here is one way to improve it.

```
#!/usr/bin/env python

print 'time to play hangman'
secret = 'crocodile'
guesses = 'aeiou'
turns = 5

while turns > 0:
  missed = 0
  for letter in secret:
    if letter in guesses:
      print letter,
    else:
      print '_',
      missed += 1

  print

  if missed == 0:
    print 'You win!'
    break

  guess = raw_input('guess a letter: ')
  guesses += guess

  if guess not in secret:
    turns -= 1
    print 'Nope.'
    print turns, 'more turns'
    if turns == 0:
      print 'The answer is', secret
```

The "missed" number counts how many blanks we are still missing. If if is zero, it means we won, and the "break" command breaks out of the "while" section early.

The "if guess not in secret" line checks if the guess was wrong. We only count down the "turns" if our guess was wrong.

When we guess wrong, we also print a bunch of stuff like "Nope" and how many more turns we have. When we are wrong for the last time we print the secret.

Making it Look Like Hangman

It will be more fun if we make our game look like Hangman.

All we need to do is print parts of the poor hangman person when there is a wrong guess. Try adding something like this to the wrong guess part:

```
...
    print 'Nope.'
    print turns, 'more turns'
    if turns < 5: print '   O   '
    if turns < 4: print ' \_|_/ '
    if turns < 3: print '   |   '
    if turns < 2: print '  / \  '
    if turns < 1: print ' d   b '
    if turns == 0:
      print 'The answer is', secret
```

You can try drawing your own person with punctuation symbols like this.

Picking a Random Secret

The only problem with the game is that it always plays the same secret word. We should use the random library to choose a random word.

Add "import random" to the top of your program and change the line that sets up the secret so that it looks like this:

```
#!/usr/bin/env python

import random

print 'time to play hangman'
secret = random.choice(['tiger', 'panda', 'mouse'])
guesses = 'aeiou'
...
```

The square brackets [ ] and commas make a list, and the random.choice picks one thing randomly from the list.

Of course, you can make the list as long as you like. Here is a longer list:

```
...
print 'time to play hangman'
secret = random.choice([
  'crocodile',
  'elephant',
  'penguin',
  'pelican',
  'leopard',
  'hamster',
])
...
```

You can write a long list on lots of lines like this, as long as you remember to end any parentheses () and brackets [] that you started. Don't forget to put a comma after each thing in the list.

Loading a List from the Internet

I have an even longer list of animals on the internet, at http://davidbau.com/data/animals.

If your computer happens to be connected to the internet, you can load this data in python using a function called "urllib.urlopen". (URLs are what web addresses are called, so urllib stands for "URL library," and urlopen means "open a URL.")

The code looks like this:

```
import urllib

animals = urllib.urlopen('http://davidbau.com/data/animals').read().split()
secret = random.choice(animals)
```

What this means is:

```
import urllib
```
Import the library called "urllib" to use.
```
animals = urllib.urlopen('http://davidbau.com/data/animals').read().split()
```
Open up the address http://davidbau.com/data/animals
```
animals = urllib.urlopen('http://davidbau.com/data/animals').read().split()
```
Read the whole webpage into a big string.
```
animals = urllib.urlopen('http://davidbau.com/data/animals').read().split()
```
Split the string into a list of words.
```
animals = urllib.urlopen('http://davidbau.com/data/animals').read().split()
```
Call the whole list "animals".
```
secret = random.choice(animals)
```
Choose a random one and call it "secret."

You can use urllib to load up more data than you could ever type in yourself.

The Whole Hangman Program

Here is the whole program from beginning to end:

```
#!/usr/bin/env python

import random
import urllib

print 'time to play hangman'
animals = urllib.urlopen('http://davidbau.com/data/animals').read().split()
secret = random.choice(animals)
guesses = 'aeiou'
turns = 5

while turns > 0:
  missed = 0
  for letter in secret:
    if letter in guesses:
      print letter,
    else:
      print '_',
      missed += 1

  print

  if missed == 0:
    print 'You win!'
    break

  guess = raw_input('guess a letter: ')
  guesses += guess

  if guess not in secret:
    turns -= 1
    print 'Nope.'
    print turns, 'more turns'
    if turns < 5: print '   O   '
    if turns < 4: print ' \_|_/ '
    if turns < 3: print '   |   '
    if turns < 2: print '  / \  '
    if turns < 1: print ' d   b '
    if turns == 0:
      print 'The answer is', secret
```

More About Programming

The best part of programming is customizing your program to make it your own.

Can you make your hangman program harder or easier?

Can you make it automatically play again at the end of the game?

This webpage at python.org has more about programming in python.