“Hangman”

Invent Your Own Computer Games with Python

Heejin Park

College of Information and Communications
Hanyang University

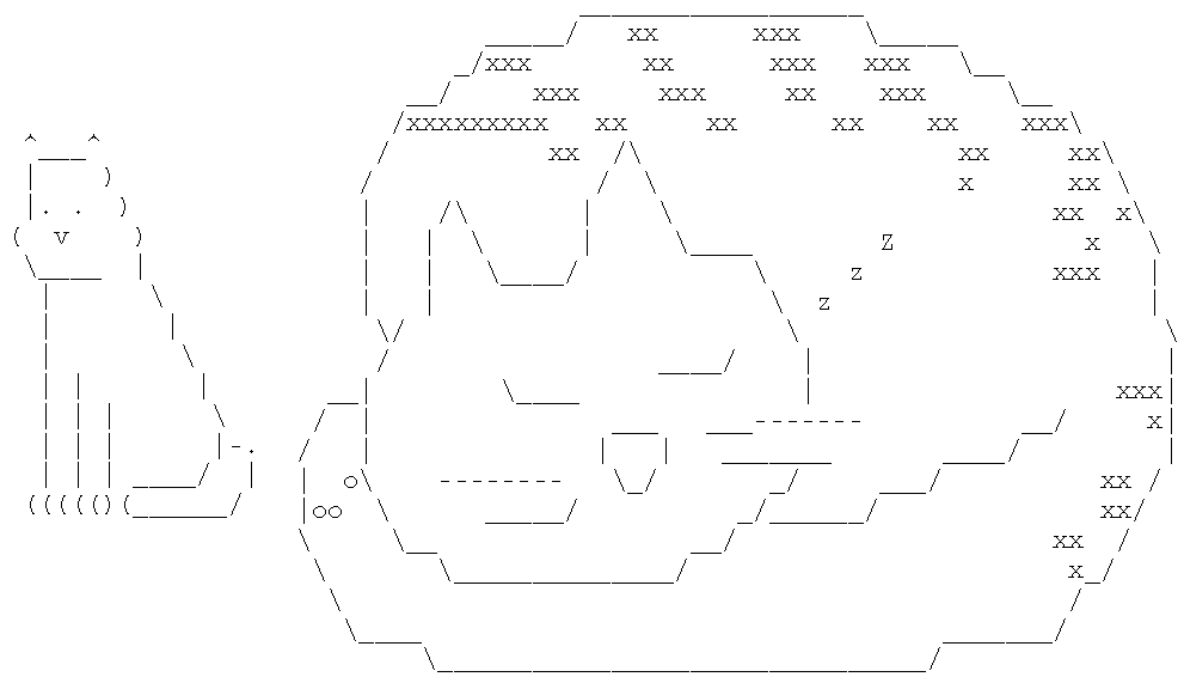
Introduction

- **ASCII Art**
- **“Hangman”**
  - Sample Run
  - Source Code
- **Designing the Program**
- **Code Explanation**
- **Things Covered In This Chapter**

ASCII Art

ASCII Art

- Half of the lines of code in the Hangman **aren't really code at all**.
- Multiline Strings that use **keyboard characters** to draw pictures.
  - **ASCII** stands for **A**merican **S**tandard **C**ode for **I**nformation **Int**



“Hangman”

Sample Run

```
H A N G M A N

  +---+
  |   |
      |
      |
      |
      |
      |
=========
Missed letters:

_ _ _
Guess a letter.
a

  +---+
  |   |
      |
      |
      |
      |
      |
=========
Missed letters:
_ a _
Guess a letter.
o

  +---+
  |   |
  O   |
      |
      |
      |
      |
=========

Missed letters: o
_ a _
Guess a letter.
r

  +---+
  |   |
  O   |
  |   |
      |
      |
      |
=========

Missed letters: or
_ a _
Guess a letter.
t

  +---+
  |   |
  O   |
  |   |
      |
      |
      |
=========
Missed letters: or
_ a t
Guess a letter.
a
You have already guessed that letter. Choose again.
Guess a letter.
c
Yes! The secret word is "cat"! You have won!
Do you want to play again? (yes or no)
no
```

“Hangman”

Source Code(1/4)

```
>>> import random
HANGMANPICS = ['''

  +---+
  |   |
      |
      |
      |
      |
=========''', '''


  +---+
  |   |
  O   |
      |
      |
      |
=========''', '''

  +---+
  |   |
  O   |
  |   |
      |
      |
=========''', '''
```

“Hangman”

Source Code(2/4)

```
  +---+
  |   |
  o   |
 /|   |
      |
      |
=========''', '''

  +---+
  |   |
  o   |
 /|\  |
      |
      |
=========''', '''

  +---+
  |   |
  o   |
 /|\  |
 /    |
      |
=========''', '''

  +---+
  |   |
  o   |
 /|\  |
 / \  |
      |
=========''']
```

“Hangman”

Source Code(3/4)

```
words = 'ant baboon badger bat bear beaver camel cat clam cobra cougar coyote crow deer dog donkey duck eagle ferret fox frog goat goose
hawk lion lizard llama mole monkey moose mouse mule newt otter owl panda parrot pigeon python rabbit ram rat raven rhino salmon seal sha
rk sheep skunk sloth snake spider stork swan tiger toad trout turkey turtle weasel whale wolf wombat zebra'.split()

def getRandomWord(wordList):
    # This function returns a random string from the passed list of strings.
    wordIndex = random.randint(0, len(wordList) - 1)
    return wordList[wordIndex]

def displayBoard(HANGMANPICS, missedLetters, correctLetters, secretWord):
    print HANGMANPICS[len(missedLetters)]
    print

    print'Missed letters:',
    for letter in missedLetters:
        print letter,
    print

    blanks = '_' * len(secretWord)

    for i in range(len(secretWord)): # replace blanks with correctly guessed letters
        if secretWord[i] in correctLetters:
            blanks = blanks[:i] + secretWord[i] + blanks[i+1:]

    for letter in blanks: # show the secret word with spaces in between each letter
        print letter,
    print

def getGuess(alreadyGuessed):
    # Returns the letter the player entered. This function makes sure the player entered a single letter, and not something else.
    while True:
        print 'Guess a letter.'
        guess = raw_input()
        guess = guess.lower()
        if len(guess) != 1:
            print 'Please enter a single letter.'
        elif guess in alreadyGuessed:
            print 'You have already guessed that letter. Choose again.'
        elif guess not in 'abcdefghijklmnopqrstuvwxyz':
            print 'Please enter a LETTER.'
        else:
            return guess
```

“Hangman”

Source Code(4/4)

```
def playAgain():
    # This function returns True if the player wants to play again, otherwise it returns False.
    print 'Do you want to play again? (yes or no)'
    return raw_input().lower().startswith('y')
print 'H A N G M A N'
missedLetters = ''
correctLetters = ''
secretWord = getRandomWord(words)
gameIsDone = False

while True:
    displayBoard(HANGMANPICS, missedLetters, correctLetters, secretWord)
    # Let the player type in a letter.
    guess = getGuess(missedLetters + correctLetters)

    if guess in secretWord:
        correctLetters = correctLetters + guess
        # Check if the player has won
        foundAllLetters = True
        for i in range(len(secretWord)):
            if secretWord[i] not in correctLetters:
                foundAllLetters = False
                break
        if foundAllLetters:
            print 'Yes! The secret word is "' + secretWord + '"! You have won!'
            gameIsDone = True
    else:
        missedLetters = missedLetters + guess

        # Check if player has guessed too many times and lost
        if len(missedLetters) == len(HANGMANPICS) - 1:
            displayBoard(HANGMANPICS, missedLetters, correctLetters, secretWord)
            print 'You have run out of guesses!\nAfter ' + str(len(missedLetters)) + ' missed guesses and ' + str(len(correctLetters)) +
' correct guesses, the word was "' + secretWord + '"'
            gameIsDone = True
    # Ask the player if they want to play again (but only if the game is done).
    if gameIsDone:
        if playAgain():
            missedLetters = ''
            correctLetters = ''
            gameIsDone = False
            secretWord = getRandomWord(words)
        else:
            break
```

Designing the Program

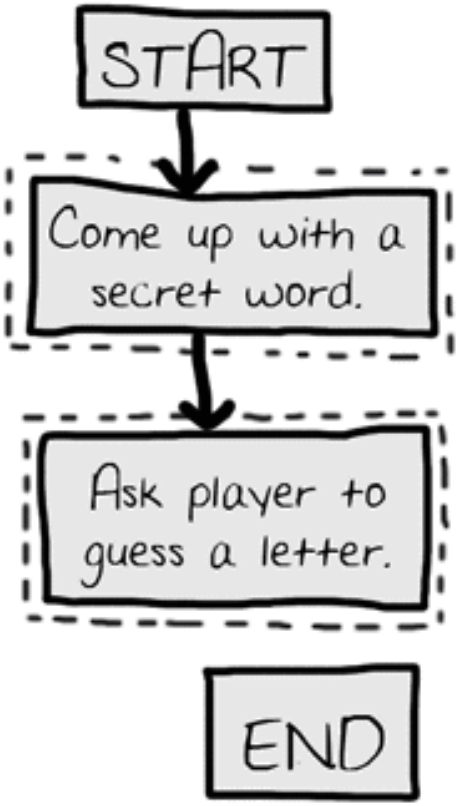
Designing a Program with a Flowchart

- Create a **flow chart** to help us visualize what this program will do.
- A flow chart is a diagram that shows a series of steps as a number of boxes connected with arrows.
- **Begin your flow chart with a <u>Start</u> and <u>End box</u>.**

Designing the Program

Designing a Program with a Flowchart

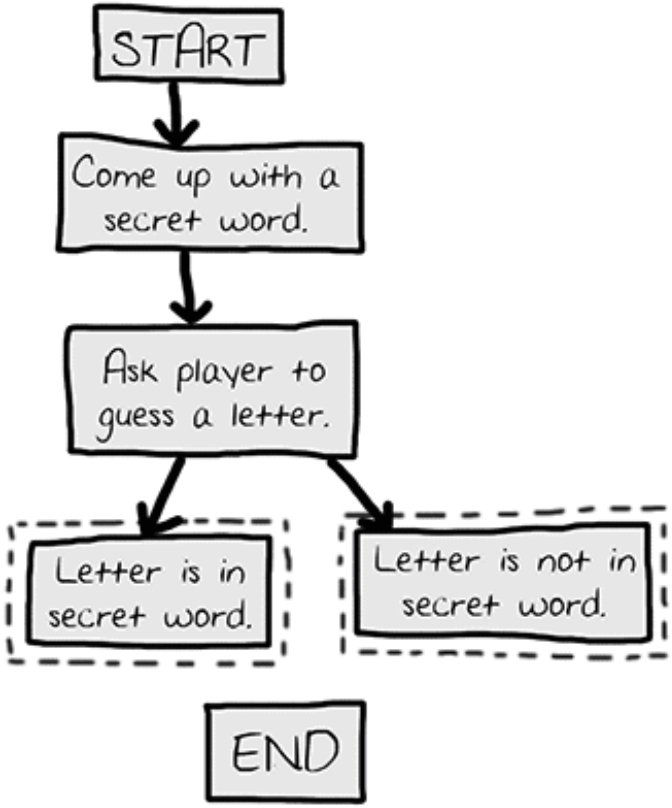
- Draw out the first **two steps** of Hangman as boxes with descriptions.



Designing the Program

Designing a Program with a Flowchart

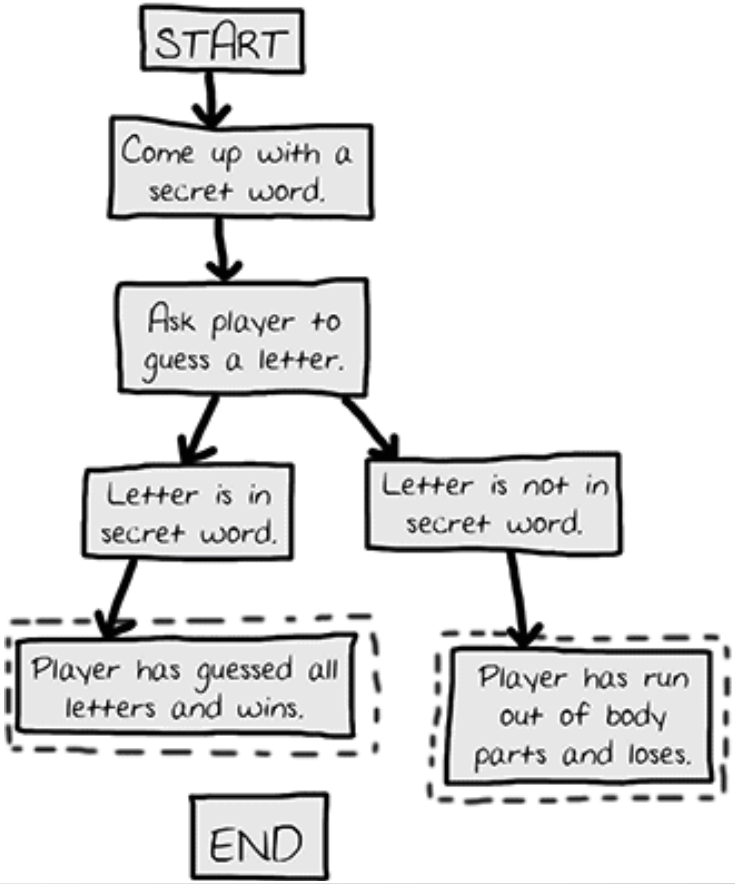
- There are **two different things** that could happen after the player guesses, so have two arrows going to separate boxes.



Designing the Program

Designing a Program with a Flowchart

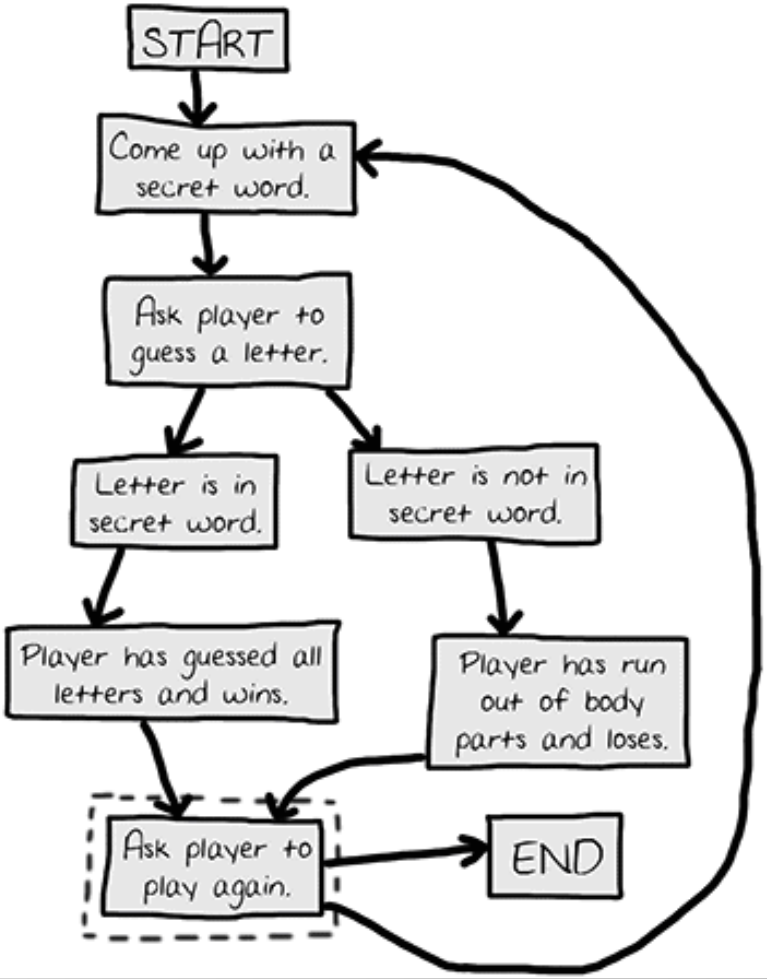
- After the branch, the steps continue on their separate paths.



Designing the Program

Designing a Program with a Flowchart

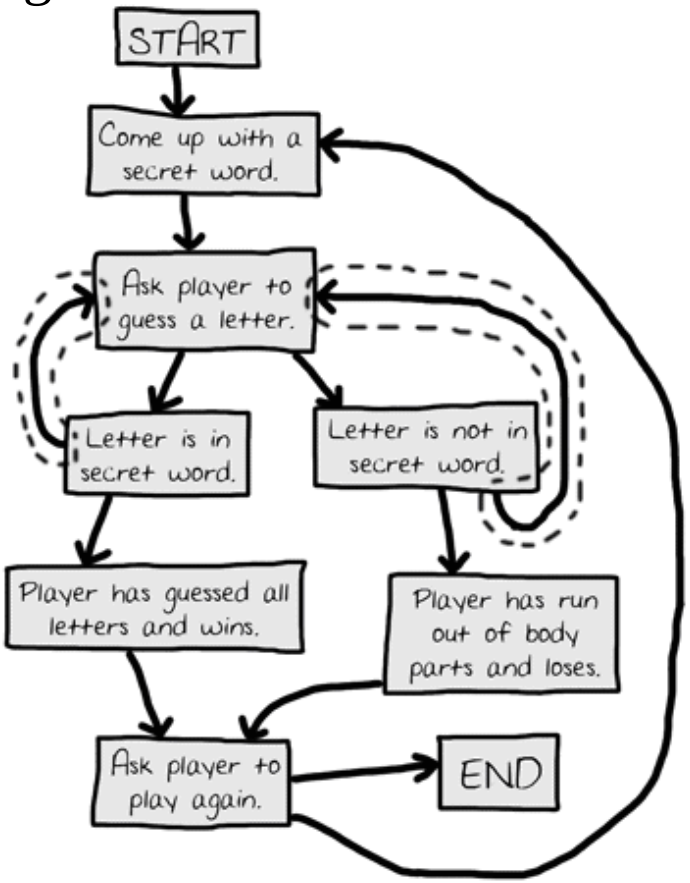
- The game ends if the player doesn't want to play again, or the game goes back to the beginning.



Designing the Program

Designing a Program with a Flowchart

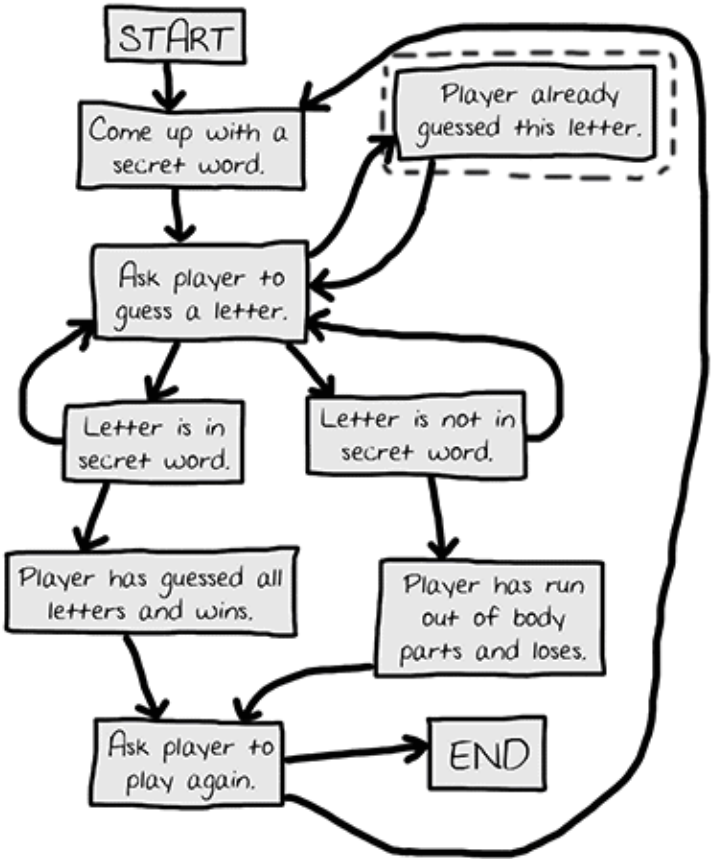
- The game does not always end after a guess. The new arrows show that the player can guess again.



Designing the Program

Designing a Program with a Flowchart

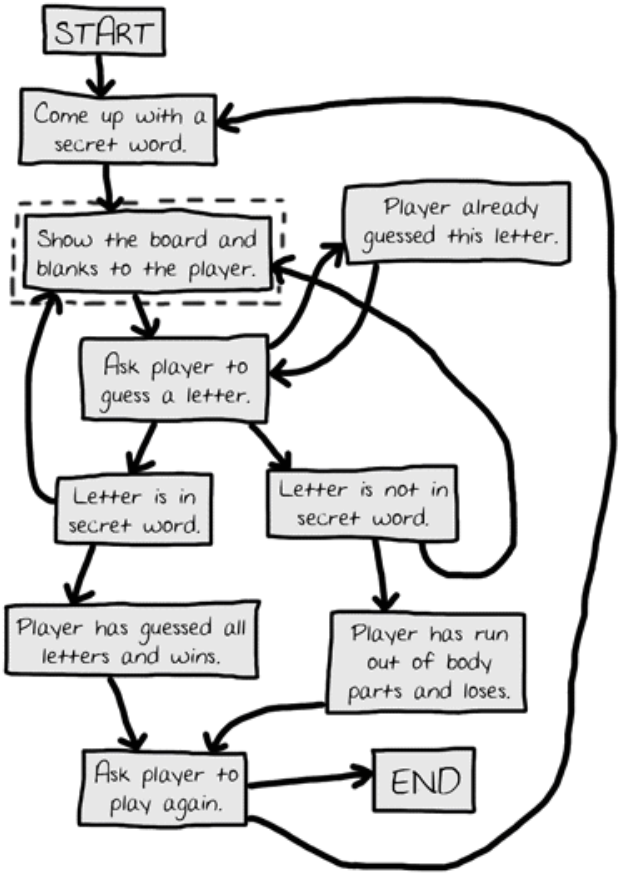
- Adding a step in case the player guesses a letter they already guessed.



Designing the Program

Designing a Program with a Flowchart

- Adding "Show the board and blanks to the player." to give the player feedback.



Code Explanation

How the Code Works

```
import random
```

- The Hangman program is going to **randomly select** a secret word from a list of secret words.
  - This means we will need the random module imported.

Code Explanation

How the Code Works

- This "line" of code is a simple variable assignment.
  - but it actually stretches over several real lines in the source code.

```
HANGMANPICS = ['''

  +---+
  |   |
      |
      |
      |
      |
=========''', '''

...the rest of the code is too big to show here...
```

Code Explanation

Multi-line Strings

- if you use **three single-quotes** instead of one single-quote to begin and end the string, the string can be on several lines.

```
>>> fizz = '''Dear Alice,
I will return home at the end of the month. I will see you then.
Your friend,
Bob'''
>>> print fizz
Dear Alice,
I will return home at the end of the month. I will see you then.
Your friend,
Bob
```

Code Explanation

Multi-line Strings

- we would have to use the **\n escape character** to represent the new lines.
  - can make the string **hard to read** in the source code.

```
>>> fizz = 'Dear Alice,\nI will return home at the end of the month.
I will see you then.\nYour friend,\nBob'
>>> print fizz
Dear Alice,
I will return home at the end of the month. I will see you then.
Your friend,
Bob
```

Code Explanation

Multi-line Strings

- Do not have to keep the same indentation to remain in the same block.
- Within the multi-line string, Python **ignores the indentation rules** it normally has for where blocks end.

```
def writeLetter():
        # inside the def-block
        print '''Dear Alice,
How are you? Write back to me soon.

Sincerely,
  Bob''' # end of the multi-line string and print statement
    print 'P.S. I miss you.' # still inside the def-block

writeLetter() # This is the first line outside the def-block.
```

Code Explanation

Constant Variables

- **HANGMANPICS**'s name is in all capitals.
  - This is the programming convention for constant variables.
  - Constants are variables whose values do not change throughout the program.

```
>>> eggs = 72
>>>
>>> DOZEN = 12
>>> eggs = DOZEN * 6
>>> eggs
72
```

Code Explanation

Lists

- A list value can contain several other values in it.
  - This is a list value that contains three string values.
  - Just like any other value, you can store this list in a variable.

```
>>> spam = ['apples', 'oranges', 'HELLO WORLD']
>>> spam
['apples', 'oranges', 'HELLO WORLD']
```

Code Explanation

Lists

- The individual values inside of a list are also called **items**.
  - **The square brackets** can also be used to get an item from a list.
  - The number between the square brackets is the **index.**

```
>>> animals = ['aardvark', 'anteater', 'antelope', 'albert']
>>> animals[0]
'aardvark'
>>> animals[1]
'anteater'
>>> animals[2]
'antelope'
>>> animals[3]
'albert'
```

Code Explanation

Lists

- Lists are very good when we have to **store lots and lots of values**.
  - but we don't want variables for each one.
  - Otherwise we would have something like this:

```
>>> animals1 = 'aardvark'
>>> animals2 = 'anteater'
>>> animals3 = 'antelope'
>>> animals4 = 'albert'
```

Code Explanation

Lists

- **Using the square brackets**
  - you can treat items in the list just like any other value.
  - the expression animals[0] + animals[2] is the same as 'aardvark' + 'antelope'.

```
>>> animals[0] + animals[2]
'aardvarkantelope'
```

Code Explanation

Quiz

- What happens if we enter an index that is larger than the list's largest index?

```
>>> animals = ['aardvark', 'anteater', 'antelope', 'albert']
>>> animals[4]
```

```
>>> animals = ['aardvark', 'anteater', 'antelope', 'albert']
>>> animals[99]
```

Code Explanation

Changing the Values of List Items with Index Assignment

- Use the square brackets to **change the value** of an item in a list.
  - overwritten with a new string.

```
>>> animals = ['aardvark', 'anteater', 'antelope', 'albert']
>>> animals[1] = 'ANTEATER'
>>> animals
['aardvark', 'ANTEATER', 'antelope', 'albert']
```

Code Explanation

List Concatenation

- **Join lists** together into one list with the + operator.
  - this is known as **list concatenation.**

```
>>> [1, 2, 3, 4] + ['apples', 'oranges'] + ['Alice', 'Bob']
[1, 2, 3, 4, 'apples', 'oranges', 'Alice', 'Bob']
```

Code Explanation

The `in` Operator

- Makes it easy to see if a value is inside a list or not.
  - Expressions that use the in operator return a **Boolean value.**
  - **True** if the value is in the list
  - **False** if the value is **not** in the list.

```
>>> animals = ['aardvark', 'anteater', 'antelope', 'albert']
>>> 'antelope' in animals
True
```

Code Explanation

- **Quiz**

```
>>> animals = ['aardvark', 'anteater', 'antelope', 'albert']
>>> 'antelope' in animals

>>> 'ant' in animals

>>> 'ant' in ['beetle', 'wasp', 'ant']
```

```
>>> 'hello' in 'Alice said hello to Bob.'
```

Code Explanation

Removing Items from Lists with `del` Statements

- You can remove items from a list with a `del` statement.

```
>>> spam = [2, 4, 6, 8, 10]
>>> del spam[1]
>>> spam
[2, 6, 8, 10]
>>> del spam[1]
>>> spam
[2, 8, 10]
>>> del spam[1]
>>> spam
[2, 10]
```

Code Explanation

Lists of Lists

- Lists are a data type that can contain other values as items in the list.
  - But these items can also be other lists.

```
>>> groceries = ['eggs', 'milk', 'soup', 'apples', 'bread']
>>> chores = ['clean', 'mow the lawn', 'go grocery shopping']
>>> favoritePies = ['apple', 'frumbleberry']
>>> listOfLists = [groceries, chores, favoritePies]
>>> listOfLists
[['eggs', 'milk', 'soup', 'apples', 'bread'], ['clean', 'mow
the lawn', 'go grocery shopping'], ['apple', 'frumbleberry']]
```

Code Explanation

Lists of Lists

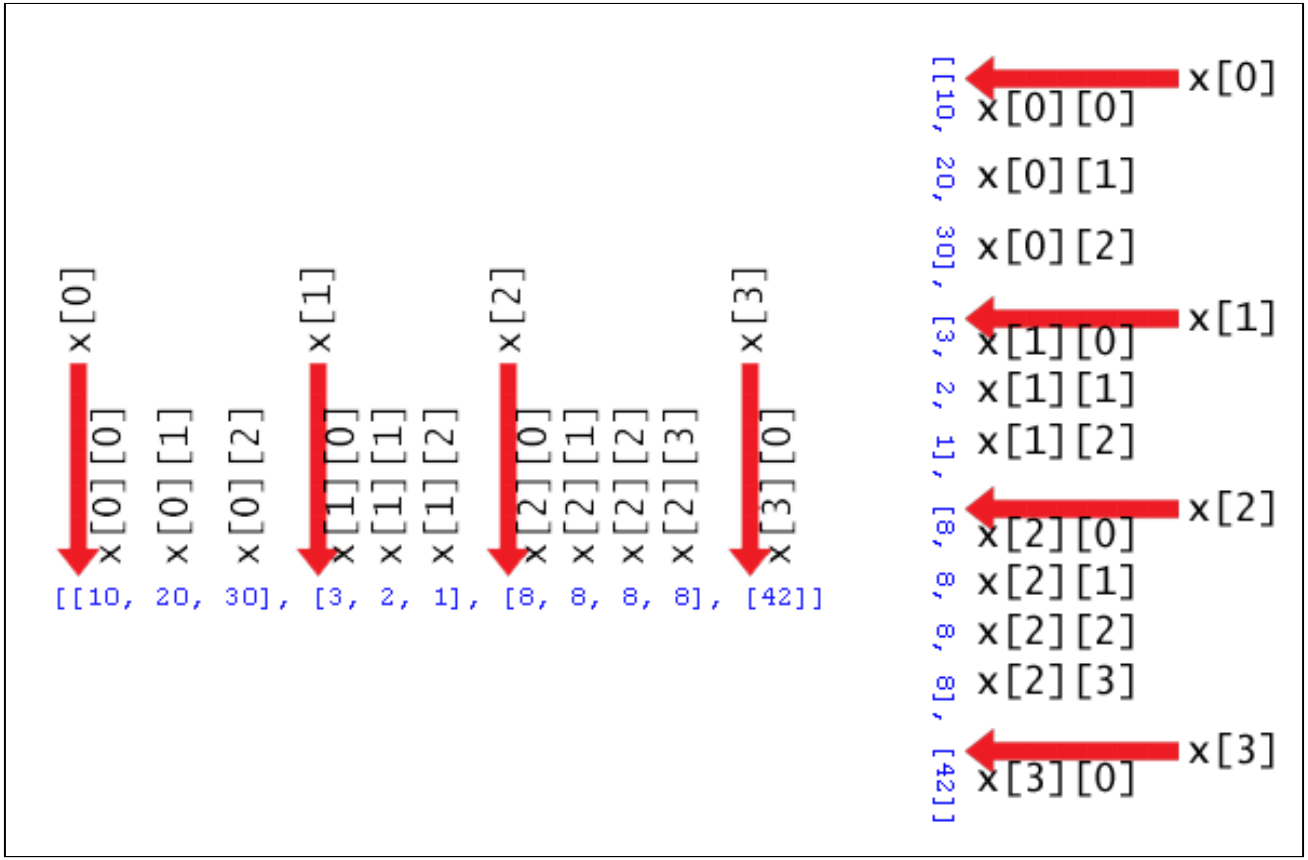
- You could also type the following and get the same values for all four variables.

```
>>> listOfLists = [['eggs', 'milk', 'soup', 'apples', 'bread']
, ['clean', 'mow the lawn', 'go grocery shopping'], ['apple',
'frumbleberry']]
>>> groceries = listOfLists[0]
>>> chores = listOfLists[1]
>>> favoritePies = listOfLists[2]
>>> groceries
['eggs', 'milk', 'soup', 'apples', 'bread']
>>> chores
['clean', 'mow the lawn', 'go grocery shopping']
>>> favoritePies
['apple', 'frumbleberry']
```

Code Explanation

Lists of Lists

- The indexes of a list of lists.



Code Explanation

List of multi-line strings

- Assign a list to the variable `words`.

```
words = 'ant baboon badger bat bear beaver camel cat
 clam cobra cougar coyote crow deer dog donkey duck
eagle ferret fox frog goat goose hawk lion lizard ll
ama mole monkey moose mouse mule newt otter owl pand
a parrot pigeon python rabbit ram rat raven rhino sa
lmon seal shark sheep skunk sloth snake spider stork
 swan tiger toad trout turkey turtle weasel whale wo
lf wombat zebra'.split()
```

Code Explanation

Methods

- Methods are just like functions, but they are always attached to a value.
- **The `lower()` and `upper()` String Methods**

```
>>> 'Hello world'.lower()
'hello world'
>>> 'Hello world'.upper()
'HELLO WORLD'
```

- **Can call a string method on that variable.**

```
>>> fizz = 'Hello world'
>>> fizz.upper()
'HELLO WORLD'
```

Code Explanation

- Quiz

```
>>> 'Hello world'.upper().lower()
```

```
>>> 'Hello world'.lower().upper()
```

Code Explanation

Methods

- **The `reverse()` List Method**
  - reverse the order of the items in the list.

```
>>> spam = [1, 2, 3, 4, 5, 6, 'meow', 'woof']
>>> spam.reverse()
>>> spam
['woof', 'meow', 6, 5, 4, 3, 2, 1]
```

Code Explanation

Methods

- **The append( ) List Method**
  - add the value you pass as an argument to the end of the list.

```
>>> eggs = []
>>> eggs.append('hovercraft')
>>> eggs
['hovercraft']
>>> eggs.append('eels')
>>> eggs
['hovercraft', 'eels']
>>> eggs.append(42)
>>> eggs
['hovercraft', 'eels', 42]
```

Code Explanation

Methods

- **The `split()` List Method**
  - This line is just one very long string, full of words separated by spaces.
  - The `split()`method changes this long string into a list, with each word making up **a single list item**.

```
words = 'ant baboon badger bat bear beaver camel cat
 clam cobra cougar coyote crow deer dog donkey duck
eagle ferret fox frog goat goose hawk lion lizard ll
ama mole monkey moose mouse mule newt otter owl pand
a parrot pigeon python rabbit ram rat raven rhino sa
lmon seal shark sheep skunk sloth snake spider stork
 swan tiger toad trout turkey turtle weasel whale wo
lf wombat zebra'.split()
```

Code Explanation

Methods

- **The `split()` List Method**
  - For an example of how the `split()` string method works.

```
>>> 'My very energetic mother just served us nine pies'.split()
['My', 'very', 'energetic', 'mother', 'just', 'served', 'us', '
nine', 'pies']
```

Code Explanation

The `len()` Function

- Takes a list as a parameter and returns the integer of how many items are in a list.

```
>>> animals = ['aardvark', 'anteater', 'antelope', 'albert']
>>> len(animals)
4
>>> people = ['Alice', 'Bob']
>>> len(people)
2
>>> len(animals) + len(people)
6
```

Code Explanation

The `len()` Function

- The square brackets by themselves are also a list value known as the **empty list.**

```
>>> len([])
0
>>> spam = []
>>> len(spam)
0
```

Code Explanation

The `getRandomWord()` Function

- store a random index for this list in the `wordIndex` variable.
- do this by calling `randint()` with two arguments.
  - The reason we need the **- 1** is because the **indexes for lists start at 0**.

```
def getRandomWord(wordList):
    # This function returns a random string from the
passed list of strings.
    wordIndex = random.randint(0, len(wordList) - 1)
    return wordList[wordIndex]
```

Code Explanation

The `displayBoard()` Function

- This function has four parameters.

```
def displayBoard(HANGMANPICS, missedLetters,
correctLetters, secretWord):
    print HANGMANPICS[len(missedLetters)]
    print
```

HANGMANPICS	a list of multi-line strings that will display the board as ASCII art
missedLetters	a string made up of the letters the player has guessed that are not in the secret word.
correctLetters	a string made up of the letters the player has guessed that are in the secret word.
secretWord	the secret word that the player is trying to guess.

Code Explanation

The `range()` Function

- When called with one argument,
  - `range()` will return a range object of integers from 0 up to the argument.

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(10000))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,...
      ...The text here has been skipped for brevity...
...9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997,
9998, 9999]
```

Code Explanation

The `range()` Function

- The list is so huge, that it won't even all fit onto the screen.
  - But we can save the list into the variable just like any other list by entering this.

```
>>> spam = list(range(10000))
```

- If you pass **two arguments** to range(),
  - the list of integers it returns is from the first argument up to the second argument.

```
>>> list(range(10, 20))
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

Code Explanation

`for` Loops

- The `for` loop is very good at looping over a list of values.
- begins with the **`for`** keyword, followed by a variable name, the **`in`** keyword, a sequence or a range object, and then a colon.
- Each time the program execution goes through the loop (on each **iteration** through the loop)

Code Explanation

for Loops

- For example

```
>>> for i in range(10):
        print i


0
1
2
3
4
5
6
7
8
9
```

Code Explanation

for Loops

- we used the for statement with the **list** instead of `range()`.

```
>>> for i in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
        print i

0
1
2
3
4
5
6
7
8
9
```

Code Explanation

- Quiz

```
for thing in ['cats', 'pasta', 'programming', 'spam']:
    print 'I really like ' + thing
```

Code Explanation

for Loops

- uses a single character from the string on each iteration.

```
>>> for i in 'Hello world!':
        print i

H
e
l
l
o

w
o
r
l
d
!
```

Code Explanation

`for` Loop

- This `for` loop will display all the missed guesses that the player has made.
- If `missedLetters` was '`ajtw`‘,
  then this for loop would display `a j t w`.

```
print'Missed letters:',
for letter in missedLetters:
    print letter,
print
```

Code Explanation

A `while` Loop Equivalent of a `for` Loop

- You can make a `while` loop that acts the same way as a `for` loop by adding extra code.

```
>>> sequence = ['cats', 'pasta', 'programming', 'spam']
>>> index = 0
>>> while (index < len(sequence)):
        thing = sequence[index]
        print 'I really like ' + thing
        index = index + 1

I really like cats
I really like pasta
I really like programming
I really like spam
```

Code Explanation

Displaying the Secret Word with Blanks

- Now we want to **print the secret word**, except we want **blank lines** for the letters.
- We can use the **_ character** (called the underscore character) for this.

secret word	blanked string
otter	_____ (five _ characters)

correctLetters	blanked string
rt	_tt_r

Code Explanation

Displaying the Secret Word with Blanks

- * operator can also be used on a string and an integer.
  - so the expression `'hello' * 3` evaluates to `'hellohellohello'`
- This will make sure that `blanks` has the same number of underscores as `secretWord` has letters.

```
blanks = '_' * len(secretWord)

for i in range(len(secretWord)): # replace blanks with correctly guessed letters
    if secretWord[i] in correctLetters:
        blanks = blanks[:i] + secretWord[i] + blanks[i+1:]

for letter in blanks: # show the secret word with spaces in between each letter
```

Code Explanation

Strings Act Like Lists

- Just think of strings as “list” of one-letter strings.

```
>>> fizz = 'Hello world!'
>>> fizz[0]
'H'
```

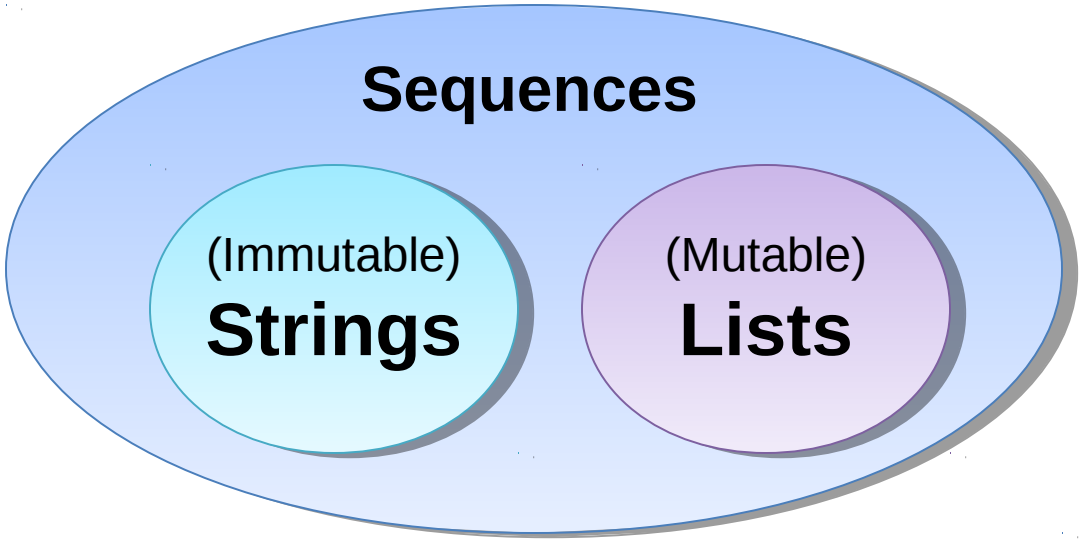
- You can also find out how many characters are in a string with the `len()` function.

```
>>> fizz = 'Hello world!'
>>> fizz[0]
'H'
>>> len(fizz)
12
```

Code Explanation

Strings Act Like Lists

- You cannot change a character in a string or remove a character with `del` statement.
  - **List: mutable sequence** (changeable)
  - **String: immutable sequence** (cannot be changed)



Code Explanation

List Slicing and Substrings

- **Slicing**
  - Like indexing with multiple indexes instead of just one.
  - Put two indexes separated by a colon.
  - Can use slicing to get a part of a string(called a **substring** from a string.)

```
>>> animals = ['aardvark', 'anteater', 'antelope', 'albert']
>>> animals[0:3]
['aardvark', 'anteater', 'antelope']
>>> animals[2:4]
['antelope', 'albert']
```

Code Explanation

List Slicing and Substrings

- **Slicing**
  - Like indexing with multiple indexes instead of just one.
  - Put two indexes separated by a colon.
  - Can use slicing to get a part of a string(called a **substring** from a string.)

```
>>> animals = ['aardvark', 'anteater', 'antelope', 'albert']
>>> animals[0:3]
['aardvark', 'anteater', 'antelope']
>>> animals[2:4]
['antelope', 'albert']
```

Code Explanation

- **Quiz**

```
>>> animals = ['aardvark', 'anteater', 'antelope', 'albert']
>>> animals[0:0]
```

```
>>> 'Hello world!'[3:8]
```

Code Explanation

Replacing the Underscores with Correctly Guessed Letters

- Let's pretend
  - the value of `secretWord` is '**otter**'
  - the value in `correctLetters` is '**tr**'
- Then `len(secretWord)` will return `5`.
- Then `range(len(secretWord))` becomes `range(5)`, which in turn returns the list `[0, 1, 2, 3, 4]`.

```
for i in range(len(secretWord)):
    if secretWord[i] in correctLetters:
        blanks = blanks[:i] + secretWord[i] + blanks[i+1:]
```

Code Explanation

Replacing the Underscores with Correctly Guessed Letters

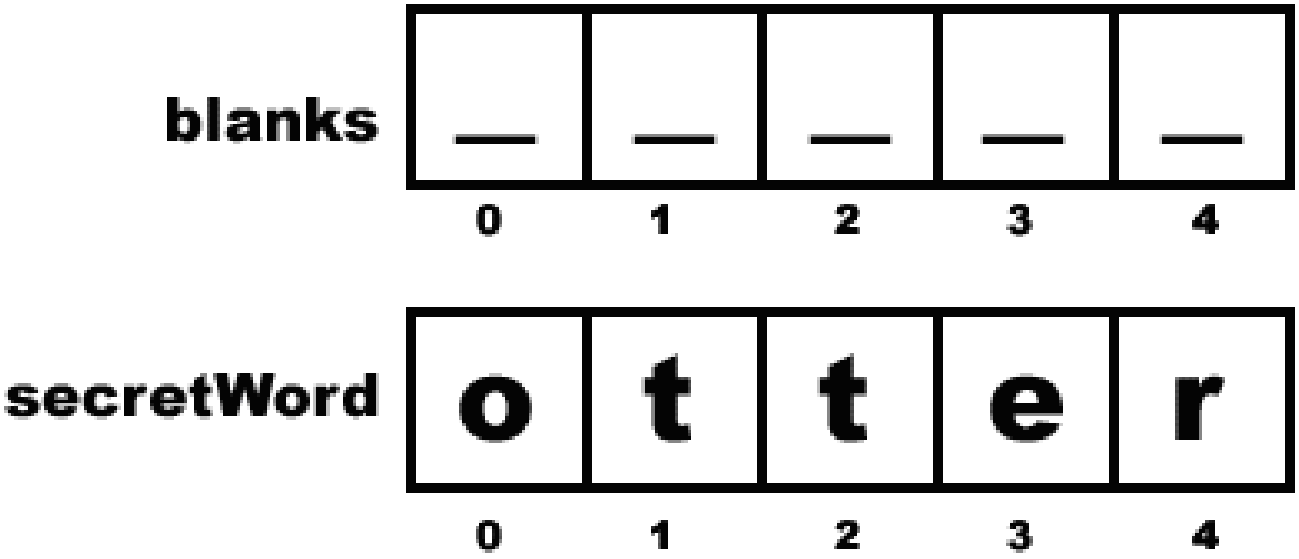
- The value of **`i`** will take on each value in **`[0, 1, 2, 3, 4]`**
  - then the **`for` loop** code is equivalent to this (called **loop unrolling)**.

```
if secretWord[0] in correctLetters:
blanks = blanks[:0] + secretWord[0] + blanks[1:]
if secretWord[1] in correctLetters:
blanks = blanks[:1] + secretWord[1] + blanks[2:]
if secretWord[2] in correctLetters:
blanks = blanks[:2] + secretWord[2] + blanks[3:]
if secretWord[3] in correctLetters:
blanks = blanks[:3] + secretWord[3] + blanks[4:]
if secretWord[4] in correctLetters:
blanks = blanks[:4] + secretWord[4] + blanks[5:]
```

Code Explanation

Replacing the Underscores with Correctly Guessed Letters

- It shows the value of the `secretWord` and `blanks` variables.
  - the index for each letter in the string.



Code Explanation

Replacing the Underscores with Correctly Guessed Letters

- The **unrolled loop** code would be the same as this.

```
if 'o' in 'tr': # False, blanks == '_____'
    blanks = '' + 'o' + '____' # This line is skipped.
if 't' in 'tr': # True, blanks == '_____'
    blanks = '_' + 't' + '___' # This line is executed.
if 't' in 'tr': # True, blanks == '_t___'
    blanks = '_t' + 't' + '__' # This line is executed.
if 'e' in 'tr': # False, blanks == '_tt__'
    blanks = '_tt' + 'e' + '_' # This line is skipped.
if 'r' in 'tr': # True, blanks == '_tt__'
    blanks = '_tt_' + 'r' + '' # This line is executed.
# blanks now has the value '_tt_r'
```

Code Explanation

Replacing the Underscores with Correctly Guessed Letters

- This `for` loop will print out each character in the string `blanks`.
- Show the secret word with spaces in between each letter

```
for letter in blanks:
    print letter,
print
```

Code Explanation

Get the Player's Guess

- The `getGuess()`
  - called whenever we want to let the player type in a letter to guess.
- `while` loop
  - it will loop forever (unless it reaches a `break` statement).
  - Such a loop is called an **infinite loop.**

```
def getGuess(alreadyGuessed):

    while True:
        print 'Guess a letter.'
        guess = raw_input()
        guess = guess.lower()
```

Code Explanation

`elif` ("Else If") Statements

- Take a look at the following code.

```
if catName == 'Fuzzball':
    print 'Your cat is fuzzy.'
else:
    print 'Your cat is not very fuzzy at all.'
```

- If the `catName` variable is equal to the string '`Fuzzball`'
  - then the `if` statement's condition is `True`
  - and we tell the user that her cat is fuzzy.
- If `catName` is anything else
  - then we tell the user her cat is not fuzzy.

Code Explanation

`elif` ("Else If") Statements

- We could put another `if` and `else` statement inside the first `else` block like this.

```
if catName == 'Fuzzball':
    print 'Your cat is fuzzy.'
else:
    if catName == 'Spots'
        print 'Your cat is spotted.'
    else:
        print 'Your cat is neither fuzzy nor spotted.'
```

Code Explanation

`elif` ("Else If") Statements

- if we wanted more things, then the code starts to have a lot of indentation.

```
if catName == 'Fuzzball':
    print 'Your cat is fuzzy.'
else:
    if catName == 'Spots'
        print 'Your cat is spotted.'
    else:
        if catName == 'FattyKitty'
            print 'Your cat is fat.'
        else:
            if catName == 'Puff'
                print 'Your cat is puffy.'
            else:
                print 'Your cat is neither fuzzy nor spotted
nor fat nor puffy.'
```

Code Explanation

`elif` ("Else If") Statements

- Using `elif`, the above code looks like this.

```
if catName == 'Fuzzball':
    print 'Your cat is fuzzy.'
elif catName == 'Spots'
    print 'Your cat is spotted.'
elif catName == 'FattyKitty'
    print 'Your cat is fat.'
elif catName == 'Puff'
    print 'Your cat is puffy.'
else:
    print 'Your cat is neither fuzzy nor spotted
nor fat nor puffy.'
```

Code Explanation

Making Sure the Player Entered a Valid Guess

- The guess variable contains the text the player typed in for their letter guess.
- The `if` statement's condition checks that the text is one and only letter.

```
if len(guess) != 1:
    print 'Please enter a single letter.'
elif guess in alreadyGuessed:
    print 'You have already guessed that letter. Choose again.'
elif guess not in 'abcdefghijklmnopqrstuvwxyz':
    print 'Please enter a LETTER.'
else:
    return guess
```

Code Explanation

Making Sure the Player Entered a Valid Guess

- The `elif` statement.

```
if len(guess) != 1:
    print('Please enter a single letter.')
elif guess in alreadyGuessed:
    print('You have already guessed that letter.')
elif guess not in 'abcdefghijklmnopqrstuvwxyz':
    print('Please enter a LETTER.')
else:
    return guess
```

One and only one of these blocks will execute.

Code Explanation

Asking the Player to Play Again

- The `playAgain()` function
  - just a `print()` function call and a return statement
  - The function call is `input()` and the method calls are `lower()` and `startswith('y')`

```
def playAgain():

    print 'Do you want to play again? (yes or no)'
    return raw_input().lower().startswith('y')
```

Code Explanation

Asking the Player to Play Again

- Here's a step by step look at how Python evaluates this expression if the user types in YES.

```
return raw_input().lower().startswith('y')
```

```
return 'YES'.lower().startswith('y')
```

```
return 'yes'.startswith('y')
```

```
return True
```

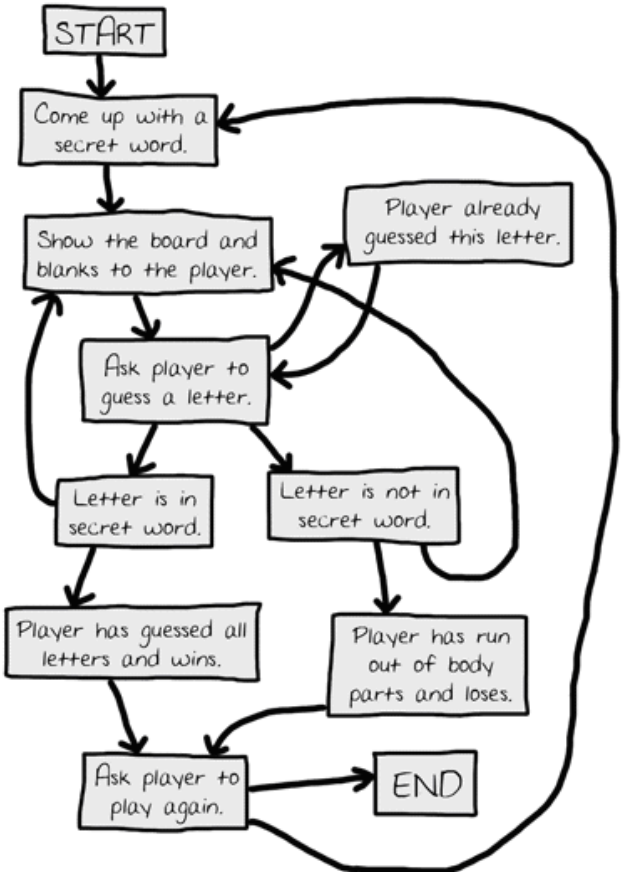
Code Explanation

- Review of the Functions We Defined
  - `getRandomWord(wordList)`
  - `displayBoard(HANGMANPICS, missedLetters, correctLetters, secretWord)`
  - `getGuess(alreadyGuessed)`
  - `playAgain()`

Code Explanation

Review of the Functions We Defined

- The complete flow chart of Hangman.



Code Explanation

The Main Code for Hangman

- **Setting Up the Variables**
  - what we do before the player starts guessing letters.
  - This line is the first actual line that executes in our game.

```
print 'H A N G M A N'
missedLetters = ''
correctLetters = ''
secretWord = getRandomWord(words)
gameIsDone = False
```

Code Explanation

Displaying the Board to the Player

- The `while` loop's condition is always `True`
  - always loop forever until a `break` statement is encountered.
  - execute a `break` statement when the game is over.

```
while True:
    displayBoard(HANGMANPICS, missedLetters,
correctLetters, secretWord)
```

Code Explanation

Letting the Player Enter Their Guess

- Remember that the function needs all the letters in `missedLetters` and `correctLetters` combined.

```
# Let the player type in a letter.
guess = getGuess(missedLetters + correctLetters)
```

Checking if the Letter is in the Secret Word

- concatenate the letter in `guess` to the `correctLetters` string

```
if guess in secretWord:
    correctLetters = correctLetters + guess
```

Code Explanation

Checking if the Player has Won

- The only way we can be sure the player won is
  - to go through each letter in `secretWord` and see if it exists in `correctLetters`.

```
# Check if the player has won
foundAllLetters = True
for i in range(len(secretWord)):
    if secretWord[i] not in correctLetters:
        foundAllLetters = False
        break
```

Code Explanation

Checking if the Player has Won

- This is a simple check to see if we found all the letters.
- If we have found every letter in the secret word
  - we should tell the player that they have won.

```
        if foundAllLetters:
            print 'Yes! The secret word is "' + secretWord +
'"! You have won!'
            gameIsDone = True
```

Code Explanation

When the Player Guesses Incorrectly

- This is the start of the `else`-block.
  - the code in this block will execute if the condition was `False`.

```
else:
```

- The player's guessed letter was wrong
  - we will add it to the `missedLetters` string.

```
missedLetters = missedLetters + guess
```

Code Explanation

When the Player Guesses Incorrectly

- How we know when the player has guessed too many times.
- Remember that each time the **player guesses wrong**,
  - **add the wrong letter** to the string in `missedLetters`.
  - the length of `missedLetters` can tell us the number of wrong guesses.

```
        # Check if player has guessed too many times and lost
        if len(missedLetters) == len(HANGMANPICS) - 1:
            displayBoard(HANGMANPICS, missedLetters, correctLetters,
secretWord)
            print 'You have run out of guesses!\nAfter ' + str(len(m
issedLetters)) + ' missed guesses and ' + str(len(correctLetters)) +
' correct guesses, the word was "' + secretWord + '"'
            gameIsDone = True
```

Code Explanation

When the Player Guesses Incorrectly

- `len(HANGMANPICS) - 1`
  - when we read the code in this program later, we know why this program behaves the way it does.
  - Of course, you could write a comment to remind yourself, like.
  - But it is easier to just use `len(HANGMANPICS) - 1` instead.

```
if len(missedLetters) == 6:
    #6 is the last index in the HANGMANPICS list
```

Code Explanation

When the Player Guesses Incorrectly

- If the player won or lost after guessing their letter
  - then our code would have set the `gameIsDone` variable to `True`.
  - If this is the case, we should ask the player if they want to play again.

```
    # Ask the player if they want to play again
(but only if the game is done).
    if gameIsDone:
        if playAgain():
            missedLetters = ''
            correctLetters = ''
            gameIsDone = False
            secretWord = getRandomWord(words)
```

Code Explanation

When the Player Guesses Incorrectly

- If the player typed in **'no'**
  - return value of the call to the `playAgain()` function would be `False`
  - the `else`-block would have executed.

```
else:
    break
```

Code Explanation

Making New Changes to the Hangman Program

- We can easily give the player more guesses
  - by **adding more multi-line strings** to the HANGMANPICS list.

```
==========''', '''

  +----+
  |    |
 [O    |
 /|\   |
 / \   |
       |
==========''', '''

  +----+
  |    |
 [O]   |
 /|\   |
 / \   |
       |
==========''']
```

Code Explanation

Making New Changes to the Hangman Program

- We can also change the list of words.
  - **colors, shapes, fruits**

```
words = 'red orange yellow green blue indigo
violet white black brown'.split()
```

```
words = 'square triangle rectangle circle ell
ipse rhombus trapazoid chevron pentagon hexag
on septagon octogon'.split()
```

```
words = 'apple orange lemon lime pear waterme
lon grape grapefruit cherry banana cantalope
mango strawberry tomato'.split()
```

Code Explanation

Dictionaries

- A collection of many values.
- Accessing the items with an index (the indexes are called **keys**) of any data type (most often **strings**).

```
>>> stuff = {'hello':'Hello there, how are you?', '
chat':'How is the weather?', 'goodbye':'It was nice
talking to you!'}
```

Code Explanation

Dictionaries

- **Curly braces { and }**
  - On the keyboard they are on the same key as the **square braces [and]**.
  - We use curly braces to type out a dictionary value in Python.
    - » The values in between them are **key-value pairs**.

```
>>> stuff['hello']
'Hello there, how are you?'
>>> stuff['chat']
'How is the weather?'
>>> stuff['goodbye']
'It was nice talking to you!'
```

Code Explanation

Getting the Size of Dictionaries with `len()`

- This will evaluate to the value for that key.
  - You can get the size with the `len()` function.

```
>>> len(stuff)
3
```

- The **list version** of this dictionary would have only the values.

```
>>> listStuff = ['Hello there, how are you?', 'How
is the weather?', 'It was nice talking to you!']
```

Code Explanation

The Difference Between Dictionaries and Lists

- Dictionaries are **unordered**.
  - Dictionaries do not have any sort of order.

```
>>> favorites1 = {'fruit':'apples', 'number':42, 'animal':'cats'}
>>> favorites2 = {'animal':'cats', 'number':42, 'fruit':'apples'}
>>> favorites1 == favorites2
True
```

Code Explanation

The Difference Between Dictionaries and Lists

- Lists are **ordered.**
  - so a list with the same values in them but in a different order are not the same.

```
>>> listFavs1 = ['apples', 'cats', 42]
>>> listFavs2 = ['cats', 42, 'apples']
>>> listFavs1 == listFavs2
False
```

Code Explanation

The Difference Between Dictionaries and Lists

- You can also use integers as the keys for dictionaries.
- Dictionaries can have keys of any data type, not just strings.

```
>>> myDict = {'0':'a string', 0:'an integer'}
>>> myDict[0]
'an integer'
>>> myDict['0']
'a string'
```

Code Explanation

The Difference Between Dictionaries and Lists

- use a dictionary in a `for` loop

```
>>> favorites = {'fruit':'apples', 'animal':'cats', 'number':42}
>>> for i in favorites:
        print i

fruit
number
animal
>>> for i in favorites:
        print favorites[i]

apples
42
cats
```

Code Explanation

The Difference Between Dictionaries and Lists

- Dictionaries also have two useful methods
  - `keys()` and `values()`
  - These will return values of a type called `dict_keys` and `dict_values`, respectively.

```
>>> favorites = {'fruit':'apples', 'animal':'cats', 'number':42}
>>> list(favorites.keys())
['fruit', 'number', 'animal']
>>> list(favorites.values())
['apples', 42, 'cats']
```

Code Explanation

Sets of Words for Hangman

- So how can we use dictionaries in our game?
  - First, let's **change the list words into a dictionary**
    - » keys are strings
    - » values are lists of strings

```
words = {'Colors':'red orange yellow green blue indigo violet white black brown'.split(),
'Shapes':'square triangle rectangle circle ellipse rhombus trapazoid chevron pentagon hexagon septagon octogon'.split(),
'Fruits':'apple orange lemon lime pear watermelon grape grapefruit cherry banana cantalope mango strawberry tomato'.split(),
'Animals':'bat bear beaver cat cougar crab deer dog donkey duck eagle fish frog goat leech lion lizard monkey moose mouse otter owl panda python rabbit rat shark sheep skunk squid tiger turkey turtle weasel whale wolf wombat zebra'.split()}
```

Code Explanation

The `random.choice()` Function

- Change our `getRandomWord()` function
  - it chooses a random word from a dictionary of lists of strings, instead of from a list of strings.
  - Here is what the function **originally** looked like:

```
def getRandomWord(wordList):
    # This function returns a random string from the
passed list of strings.
    wordIndex = random.randint(0, len(wordList) - 1)
    return wordList[wordIndex]
```

Code Explanation

The `random.choice()` Function

- Change our `getRandomWord()` function
- **Change** the code in this function so that it looks like this:

```
def getRandomWord(wordDict):
    # This function returns a random string from the passed
dictionary of lists of strings, and the key also.
    # First, randomly select a key from the dictionary:
    wordKey = random.choice(list(wordDict.keys()))

    # Second, randomly select a word from the key's list in
the dictionary:
    wordIndex = random.randint(0, len(wordDict[wordKey]) - 1)

    return [wordDict[wordKey][wordIndex], wordKey]
```

Code Explanation

The `random.choice()` Function

- **`randint(a, b)`**
  - return a random integer between the two integers `a` and `b`
  - **`choice(a)`** returns a random item from the list `a`

```
>>> random.randint(0, 9)
>>> random.choice(list(range(0, 10)))
```

Code Explanation

Evaluating a Dictionary of Lists

- `wordDict[wordKey][wordIndex]` may look kind of complicated
- but it is just an expression you can evaluate one step at a time like anything else.

```
wordDict[wordKey][wordIndex]
        ↓
wordDict['Fruits'][5]
        ↓
['apple', 'orange', 'lemon', 'lime', 'pear', 'watermelon'
, 'grape', 'grapefruit', 'cherry', 'banana', 'cantalope',
'mango', 'strawberry', 'tomato'][5]
        ↓
'watermelon'
```

Code Explanation

Evaluating a Dictionary of Lists

- There are just **three more changes** to make to our program.
  - **The first two** are on the lines that we call the **`getRandomWord()`** function.
  - The function is called on lines 148 and 184 in the original program

```
147.            correctLetters = ''
148.            secretWord = getRandomWord(words)
149.            gameIsDone = False
...

183.            gameIsDone = False
184.            secretWord = getRandomWord(words)
185.        else:
```

Code Explanation

Evaluating a Dictionary of Lists

- We would then have to change the code as follows

```
147. correctLetters = ''
148. secretWord = getRandomWord(words)
149. secretKey = secretWord[1]
150. secretWord = secretWord[0]
151. gameIsDone = False
...

182.                gameIsDone = False
183.                secretWord = getRandomWord(words)
184.                secretKey = secretWord[1]
185.                secretWord = secretWord[0]
186.            else:
```

Code Explanation

Multiple Assignment

- An easier way by doing a little trick with assignment statements.
  - to put the same number of variables on the left side of the = sign as are in the list on the right side of the = sign.

```
>>> a, b, c = ['apples', 'cats', 42]
>>> a
'apples'
>>> b
'cats'
>>> c
42
```

Code Explanation

- **Quiz**

```
>>> a, b, c, d = ['apples', 'cats', 42]
```

```
>>> a, b, c, d = ['apples', 'cats']
```

Code Explanation

Multiple Assignment

- So we should change our code in Hangman to use this trick
  - which will mean our program uses fewer lines of code.

```
147. correctLetters = ''
148. secretWord, secretKey = getRandomWord(words)
149. gameIsDone = False
...

182.                gameIsDone = False
183.                secretWord, secretKey = getRandomWord
     (words)
184.            else:
```

Code Explanation

Printing the Word Category for the Player

- **The last change**
  - to add a simple print statement to tell the player which set of words they are trying to guess.
  - Here is the original code:

```
151. while True:
152.     displayBoard(HANGMANPICS, missedLetters,
correctLetters, secretWord)
```

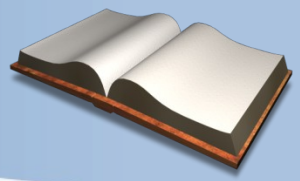
Code Explanation

Printing the Word Category for the Player

- **The last change**
  - Add the line so your program looks like this:

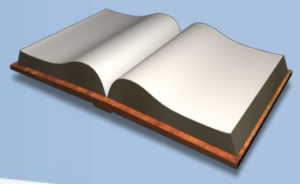
```
151. while True:
152.     print 'The secret word is in the set: ' +
secretKey
153.     displayBoard(HANGMANPICS, missedLetters,
correctLetters, secretWord)
```

Things Covered In This Chapter(1/3)



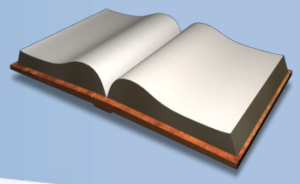
- Designing our game by drawing a flow chart before programming.
- ASCII Art
- Multi-line Strings
- Lists
- List indexes
- Index assignment
- List concatenation
- The `in` operator
- The `del` operator
- Methods
- The `append()` list method

Things Covered In This Chapter(2/3)



- The `lower()` and `upper()` string methods
- The `reverse()`list method
- The `split()`list method
- The `len()`function
- Empty lists
- The `range()` function
- `for` loops
- Strings act like lists
- Mutable sequences(lists) and immutable sequences(strings)
- List slicing and substrings
- `elif` statements

Things Covered In This Chapter(3/3)



- The `startswith(someString)` and `endswith(someString)`string methods
- The dictionary data type(which is unordered, unlike list data type which is ordered)
- key-value pairs
- The `keys()`and `values()` dictionary methods.
- Multiple variable assignment, such as `a, b, c = [1, 2, 3]`