

Python Games



Session 2

By Declan Fox



Rules

“Above all, be cool.”

General Information

Wi-Fi Name: CoderDojo

Password: coderdojowireless

Website: <http://cdathenry.wordpress.com/>

Recommended reading:

<http://inventwithpython.com>

Social Media

Like our new Facebook page at
www.facebook.com/CoderDojoAthenry

Or if you are on twitter follow us on
[@coderdojoathenr](https://twitter.com/coderdojoathenr)

Installation

As we will be moving on to graphical games we will need to install both Python and Pygame*

* If you have Python 3.x.x and Pygame installed you can ignore the next slide

Installation

We are using version 3.2 of Python go to <https://www.python.org/download/releases/3.2.5/>

Select [Windows x86 MSI Installer \(3.2.5\)](#)

To install Pygame go to

<http://pygame.org/download.shtml>

Select [pygame-1.9.2a0.win32-py3.2.msi](#)

Last week

```
while_if_break.py - C:/Users/coderdojo/Documents/2014/Day 2/while_if_break.py
File Edit Format Run Options Windows Help
#This program revisits last weeks code.

import random

total = random.randint(1,10)
number = 0

while total <= 100:
    print('Total = ' + str(total) + ' enter a number to add to it')
    number = int(input('or enter 0 to quit '))
    if number == 0:
        print('You entered 0')
        break

    total = total + number

if total > 100:
    print('The total is greater than 100')

print('Goodbye')
```

Last Week

1. Variables
2. Print function
3. Input function
4. Import statement
5. Random
6. Function arguments
7. While loop

Last Week

8. Code blocks
9. Boolean data
10. Comparison operators
11. Conditions
12. int function
13. If statement
14. Break statement

My guessing game

```
warmCold.py - C:/Users/coderdojo/Documents/2014/Day 2/warmCold.py
File Edit Format Run Options Windows Help
# This is a guess the number game.
import random

guessesTaken = 0
lastGuess = 10

print('Hello! What is your name?')
myName = input()

number = random.randint(1, 20)
print('Well, ' + myName + ', I am thinking of a number between 1 and 20.')

while guessesTaken < 10:
    print('Take a guess.')
    guess = input()
    guess = int(guess)

    guessesTaken = guessesTaken + 1

    if guess == number:
        break

    oldDiff = number - lastGuess
    if oldDiff < 0:
        oldDiff = oldDiff * -1

    newDiff = number - guess
    if newDiff < 0:
        newDiff = newDiff * -1

    if newDiff < oldDiff:
        print('Warmer')

    if newDiff > oldDiff:
        print('colder')
```

Ln: 1

Dragons Realm

Text based adventure game

New Concepts

- Escape characters
- The end keyword
- Functions
- Boolean operators

Escape Characters

Suppose we want to print out the following:

John's house

John said "Hello"

green\teal

Old\nnew

Escape Characters

John's house

Python thinks that the single quote means the end of the string.

If we use the single quote escape character `\'` this will tell python to print the single quote.

```
print ('john\'s house')
```

Escape Characters

Escape Characters	
Escape Character	What Is Actually Printed
<code>\\</code>	Backslash (\)
<code>\'</code>	Single quote (')
<code>\"</code>	Double quote (")
<code>\n</code>	Newline
<code>\t</code>	Tab

The end keyword

Normally, `print()` adds a newline character to the end of the string it prints. (This is why a blank `print()` function will just print a newline.)

But the `print()` function can optionally have a second parameter (called `end`.)

```
print('hello', end='')
```

```
print(' world')
```

FUNCTIONS

Why are functions important?

- **The program is easier to understand.**
 - Someone reading it can choose whether or not to look at the details of a function.
- **You can reuse code easily.**
 - Instead of copying and pasting code, put it in a function and call it more than once. If you need to change the code, you change it in one place.
- **Easier to design and test code.**

Functions in Software Design

Making software involves continually shifting between top-down design and building bottom-up

- **Top-down design**-- Take the whole program and break it down into parts.
- **Build from the bottom up** -- Code and test a particular function, starting at the bottom.

What can a Function do?

1. It can perform some computational task.
2. It can return a result and/or modify parameters.
3. It can take in Data (called Arguments)

Defining a Function

The **def** statement

```
def displayIntro():
```

The `def` statement is made up of the `def` keyword, followed by a function name with parentheses, and then a colon (the `:` sign). There is a block after the statement called the `def`-block. The code in this block is executed when we call the `displayIntro()` function later in the program.

Return Values

`return` cave

This is the return keyword, which only appears inside def-blocks. Remember how the `input()` function returns the string value that the player typed in? Or how the `randint()` function will return a random integer value? Our function will also return a value. It returns the string that is stored in `cave`.

Function Variables

Just like the values in our program's variables are forgotten after the program ends, variables created inside a function are forgotten after the execution leaves the function. Not only that, but when execution is inside the function, we cannot change the variables outside of the function, or variables inside other functions.

Local and Global Variables

- Variables defined inside a function are called **Local Variables** and can only be read or modified inside that function.
- Variables defined outside of a function are called **Global Variables** and can be read outside and inside functions, but can only be modified outside of all functions.

Parameters

Parameters are local variables that get defined when a function is called. The value stored in the parameter is the argument that was passed in the function call.

Parameters

For example, here is a short program that demonstrates parameters.

```
def adder(x,y):
```

```
    z = x + y
```

```
    return z
```

```
firstNumber = 3
```

```
secondNumber = 5
```

```
answer = adder(firstNumber,secondNumber)
```

```
print(answer)
```

Where to Put Function Definitions

A function's definition (where we put the def statement and the def-block) has to come before you call the function.

This is like how you must assign a value to a variable before you can use the variable. If you put the function call before the function definition, you will get an error.

Function calls

We call our own functions the same way we call Python's built in functions.

Example- myFunction ()

A function can be called from anywhere once it's defined.

For example functions can be called from the main part of the program, from another function even from inside itself(example countdown).

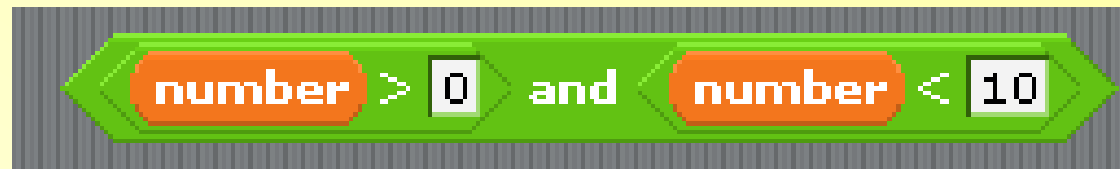
Boolean Operators

The and operator

Boolean logic deals with things that are either true or false. This is why the Boolean data type only has two values, True and False.

We can use the **and** Boolean operator to combine two Boolean values to produce a new Boolean value.

while number > 0 **and** number < 10:



Boolean Operators

Think of the sentence, "Cats have whiskers and dogs have tails." This sentence is true, because "cats have whiskers" is true and "dogs have tails" is also true.

But the sentence, "Cats have whiskers and dogs have wings" would be false. Even though "cats have whiskers" is true, dogs do not have wings, so "dogs have wings" is false. The entire sentence is only true if both parts are true because the two parts are connected by the word "and."

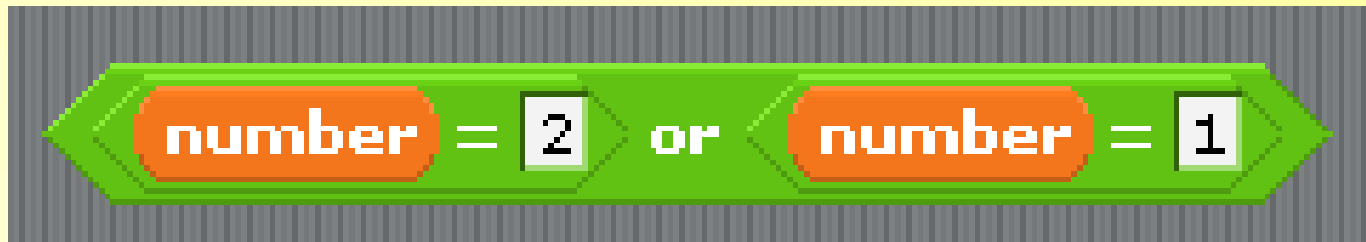
If one or both parts are false, then the entire sentence is false.

Boolean Operators

The or operator

The or operator works similar to the and, except it will evaluate to True if either or both of the two Boolean values are True. The only time the or operator evaluates to False is if both of the Boolean values are False.

`while number == 1 or number == 2:`



Boolean Operators

The sentence "Cats have whiskers or dogs have wings." is true. Even though dogs don't have wings, when we say "or" we mean that one of the two parts is true. The sentence "Cats have whiskers or dogs have tails." is also true.

Most of the time when we say "this or that", we mean one thing is true but the other thing is false. In programming, "or" means that either of the things are true, or maybe both of the things are true.

Boolean Operators

The not Operator

The not operator only works on one value. There is only value on the right side of the not keyword.

The not operator will evaluate to True as False and will evaluate False as True.

`while (not number < 1) and number < 10:`



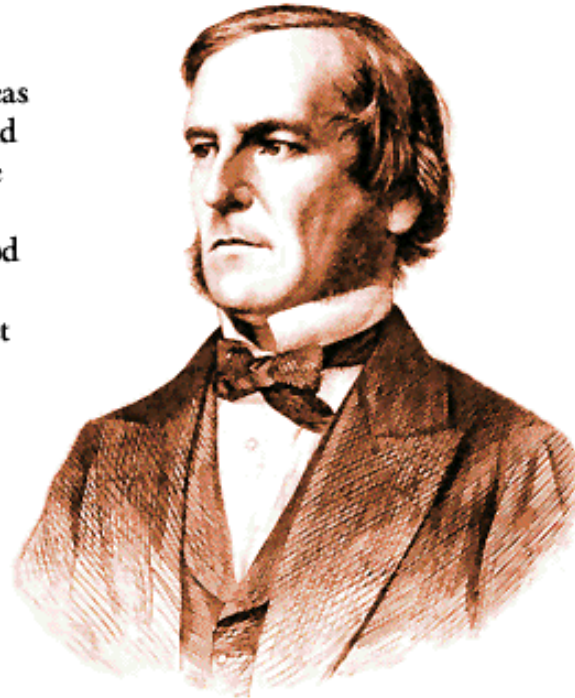
Boolean Operators

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	(a and b) is true.
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(a or b) is true.
not	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	not(a and b) is false.

Boolean Operators

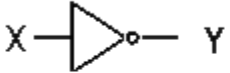


GEORGE BOOLE (1815–64)

THE SELF-TAUGHT English mathematician George Boole produced important work in many areas of the discipline, including **calculus** and the theory of **probability**. However, he is chiefly remembered as a pioneer of **formal logic**. Boole developed a method of reducing statements in logic to algebraic statements, using a simple set of symbols. Although logicians have since streamlined the set of symbols, Boole's basic system survives. An interpretation of **Boolean algebra** in terms of **truth values**, called the **propositional calculus**, forms the basis of the digital processes in modern computers.

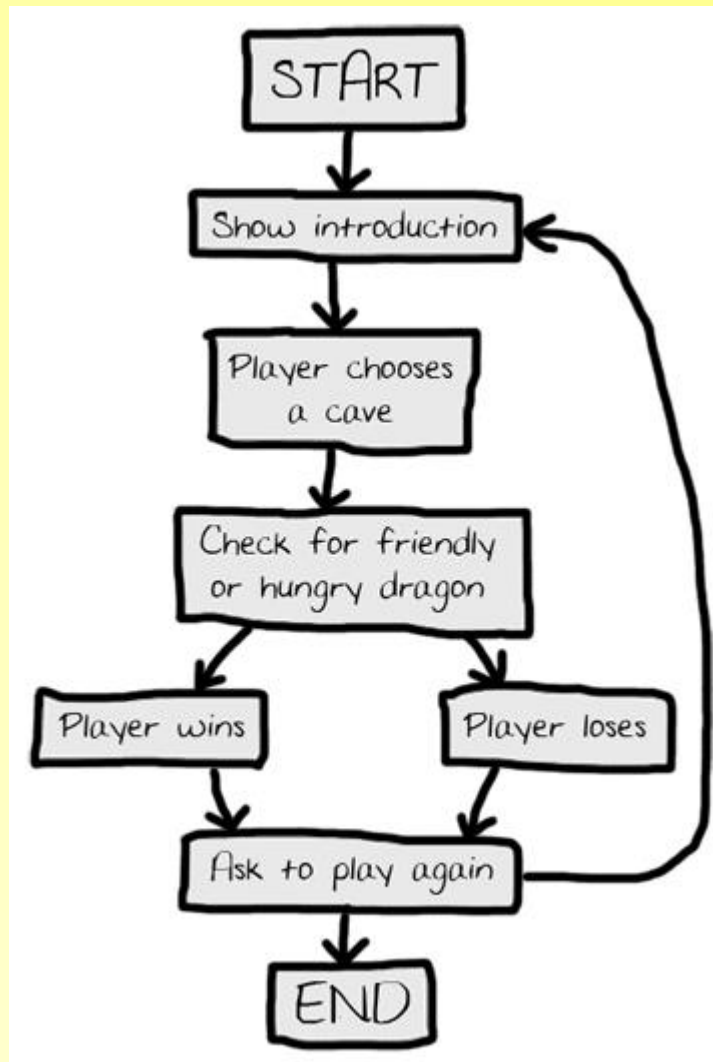


Digital Electronics

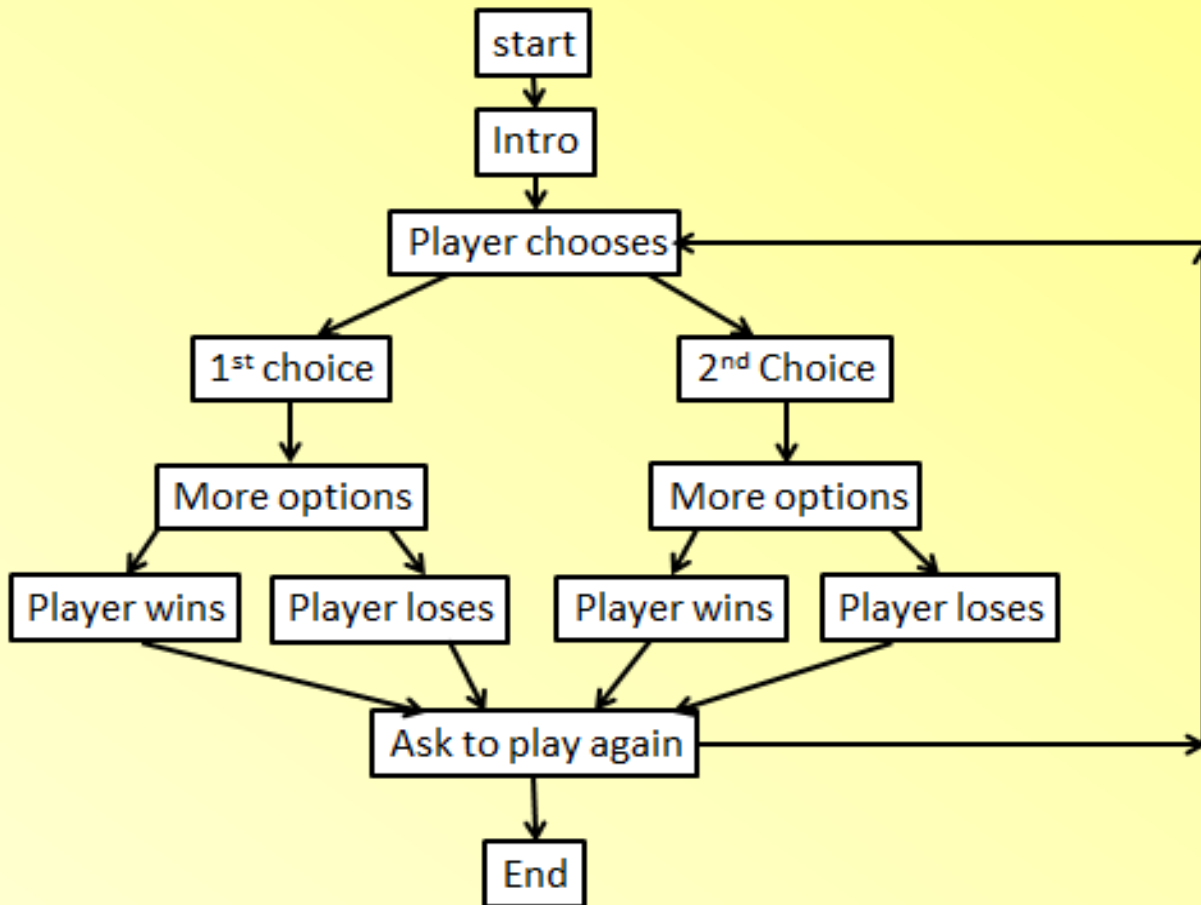
Logic Gates

NOT	\bar{X}		<table border="1"><thead><tr><th>X</th><th>Y</th></tr></thead><tbody><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></tbody></table>	X	Y	0	1	1	0									
X	Y																	
0	1																	
1	0																	
AND	$X \cdot Y$		<table border="1"><thead><tr><th>X</th><th>Y</th><th>Z</th></tr></thead><tbody><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></tbody></table>	X	Y	Z	0	0	0	0	1	0	1	0	0	1	1	1
X	Y	Z																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR	$X + Y$		<table border="1"><thead><tr><th>X</th><th>Y</th><th>Z</th></tr></thead><tbody><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></tbody></table>	X	Y	Z	0	0	0	0	1	1	1	0	1	1	1	1
X	Y	Z																
0	0	0																
0	1	1																
1	0	1																
1	1	1																

Our Own Game



Our own game



Next Week

We'll review functions then we'll continue working on our Adventure Game