Python

Session 3

By Declan Fox
With thanks to Al Sweigart

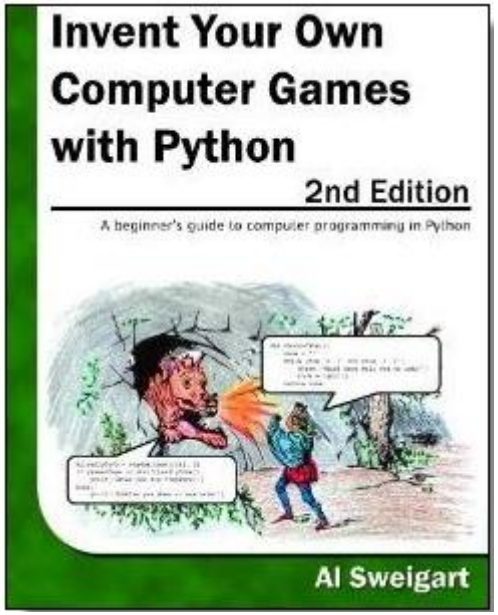
“Above all, be cool.”

Wi-Fi Name: CoderDojo

Password: coderdojowireless

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

Remember to check out “Invent your own computer games with Python” by Al Sweigart.



The book is free online at

http://inventwithpython.com

Next Week

No Coderdojo next week Saturday 26th October

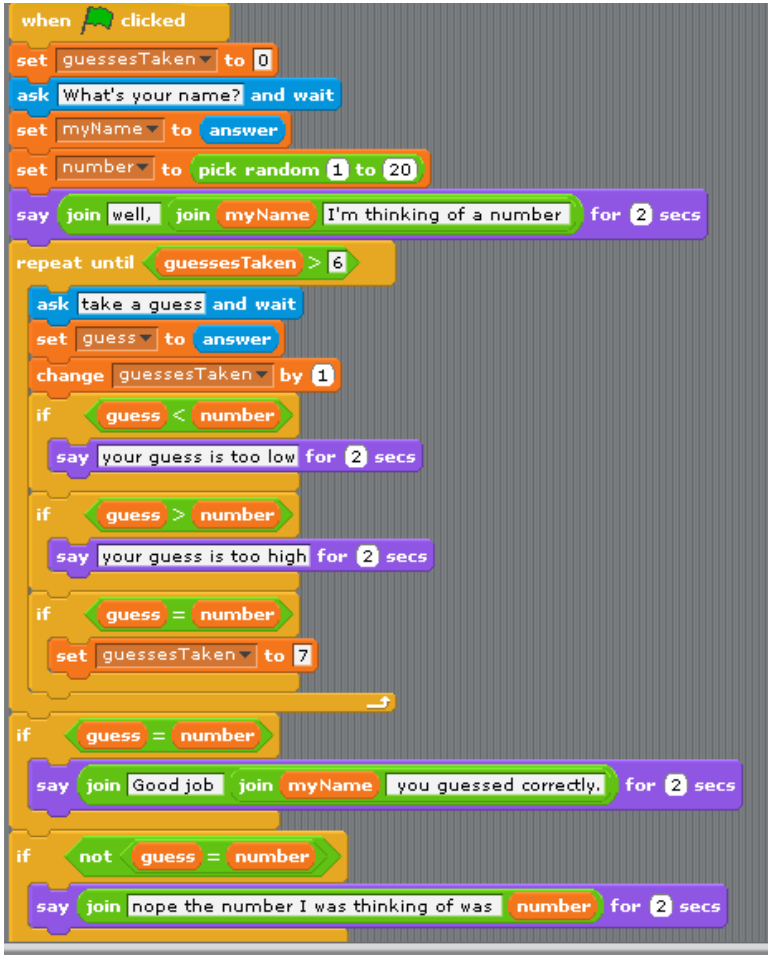
We will return on Saturday 2nd November

guess.py - C:\Users\User\Documents\python session1\guess.py

File Edit Format Run Options Windows Help

```
# This is a guess the number game.
import random

guessesTaken = 0

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 < 6:
    print('Take a guess.') # There are four spaces in front of print.
    guess = input()
    guess = int(guess)

    guessesTaken = guessesTaken + 1

    if guess < number:
        print('Your guess is too low.') # There are eight spaces in front of print.

    if guess > number:
        print('Your guess is too high.')

    if guess == number:
        break

if guess == number:
    guessesTaken = str(guessesTaken)
    print('Good job, ' + myName + '! You guessed my number in ' + guessesTaken + ' guesses!')

if guess != number:
    number = str(number)
    print('Nope. The number I was thinking of was ' + number)
```

Ln: 1 Col: 0


when clicked
set guessesTaken to 0
ask What's your name? and wait
set myName to answer
set number to pick random 1 to 20
say join well, join myName I'm thinking of a number for 2 secs
repeat until guessesTaken > 6
ask take a guess and wait
set guess to answer
change guessesTaken by 1
if guess < number
say your guess is too low for 2 secs
if guess > number
say your guess is too high for 2 secs
if guess = number
set guessesTaken to 7
if guess = number
say join Good job join myName you guessed correctly. for 2 secs
if not guess = number
say join nope the number I was thinking of was number for 2 secs

Import Statement

```
import random
```

This line is an **import statement**

Python includes many built-in functions.

Modules are Python programs that contain additional functions.

In this case, we're importing the random module.

Random

number = random.randint(1, 20)

In this line we call a new function named randint(), and then store the return value in a variable named number.

Because the randint() function is provided by the random module, we precede it with random. (don't forget the dot) to tell our program that the function randint() is in the random module.

Arguments

Passing Arguments to Functions

The numbers in the parentheses in the random.randint(1, 20) function call are called arguments.

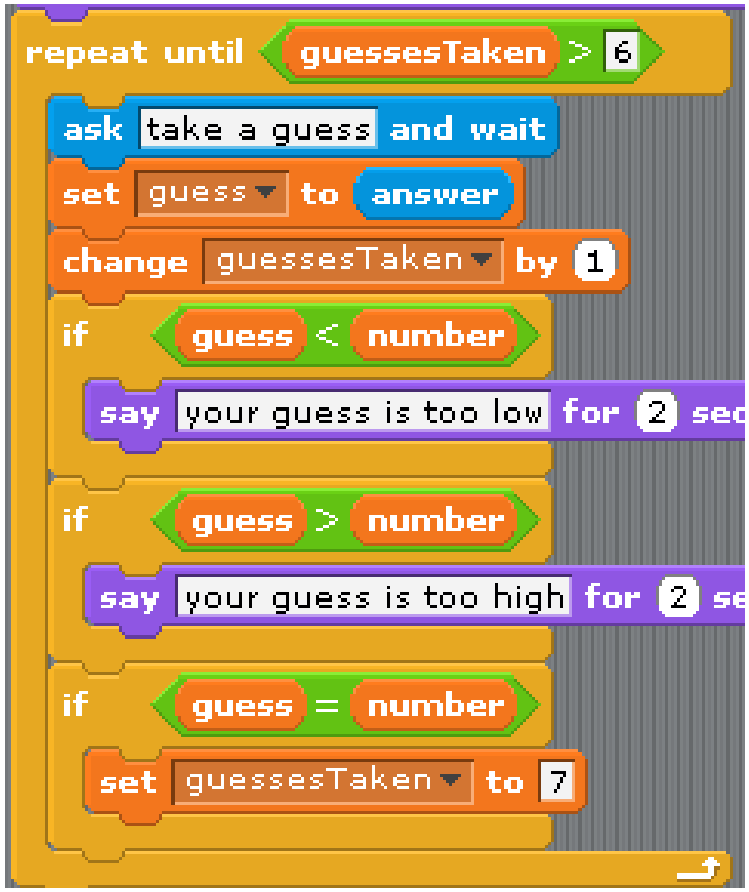
Arguments are the values that are passed to a function when the function is called. Arguments tell the function how to behave.

Some functions require that you pass them values when you call them.

loops

Line 12 has something called a while statement, which indicates the beginning of a while loop.

Loops are parts of code that are executed over and over again.


repeat until guessesTaken > 6
ask take a guess and wait
set guess to answer
change guessesTaken by 1
if guess < number
say your guess is too low for 2 sec
if guess > number
say your guess is too high for 2 se
if guess = number
set guessesTaken to 7

Code Blocks

- A **block** is one or more lines of code grouped together with the same amount of indentation. You can tell where a block begins and ends by looking at the line's **indentation** (that is, the number of spaces in front of the line).
- A block begins when a line is indented by four spaces. Any following line that is also indented by four spaces is part of the block. A block within a block begins when a line is indented with another four spaces (for a total of eight spaces in front of the line). The block ends when there is a line of code with the same indentation before the block started.

Blocks

```
while guessesTaken < 6:
    print('Take a guess.') # There are four spaces in front of print.
    guess = input()
    guess = int(guess)

    guessesTaken = guessesTaken + 1

    if guess < number:
        print('Your guess is too low.') # There are eight spaces in front of print.

    if guess > number:
        print('Your guess is too high.')

    if guess == number:
        break
```

The Boolean Data Type

The Boolean data type has only two values:

True or False.

These values are case-sensitive and they are not string values; in other words, you do not put a ' quote character around them.

We will use Boolean values (also called bools) with comparison operators to form conditions.

Comparison Operators

while guessesTaken < 6:

The comparison operator is used to compare two values and evaluate to a True or False Boolean value.

Comparison Operators

Comparison operators	
Operator Sign	**Operator Name**
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Is equal to
!=	Not equal to

Conditions

A condition is an expression that combines two values with a comparison operator (such as < or >) and evaluates to a Boolean value. A condition is just another name for an expression that evaluates to True or False.

Conditions always evaluate to a Boolean value: True or False.

In the case of our Guess the Number program, in line 4 we stored the value 0 in guessesTaken. Because 0 is less than 6, this condition evaluates to the Boolean value of True.

A condition is just a name for an expression that uses comparison operators such as < or !=.

Converting Strings to Integers with the int() Function

guess = int(guess)

In line 15, we call a new function called int().

The input() function returned a string of text that player typed. But in our program, we will want an integer, not a string. When the player enters their guess, the input() function will return the string value '5' and not the integer value 5. Remember that Python considers the string '5' and the integer 5 to be different values.

So the int() function will convert the string value we give it and return the integer value of it.

Incrementing Variable

guessesTaken = guessesTaken + 1

This increases the value of guessesTaken by 1

if Statements

Lines 19 and 20 check if the number that the player guessed is less than the secret random number that the computer came up with. If so, then we want to tell the player that their guess was too low by printing this message to the screen.

```
if guess < number:
    print('Your guess is too low.')
```

Leaving Loops Early with the break Statement

```
if guess == number:
    break
```

The line inside the if-block is a **break** statement that tells the program to immediately jump out of the while-block to the first line after the end of the while-block.

Escape Characters

Suppose we want to print out the following:

John’s house

John said “Hello”

green\teal

Old\new

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**
\\	Backslash (\)
\'	Single quote (')
\"	Double quote (")
\n	Newline
\t	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.

Global and Local Variables

- 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.
- Variables defined inside a function are called **Local Variables** and can only be read or modified inside that function.

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.

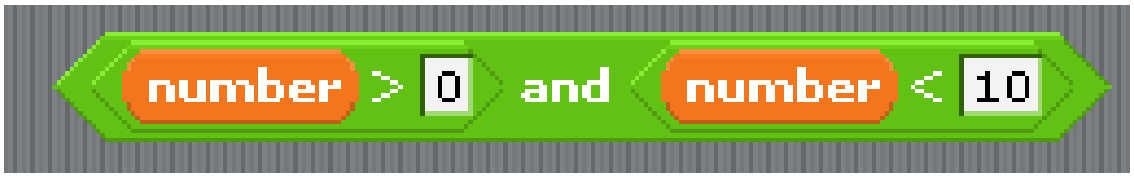
For example functions can be called from the main part of the program, from another function even from inside itself(example countdown).

Boolean Operators

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.“

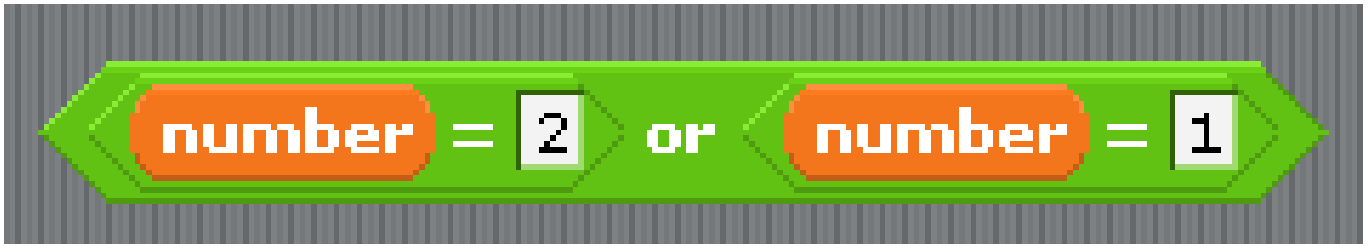
If one or both parts are false, then the entire sentence is false.

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.

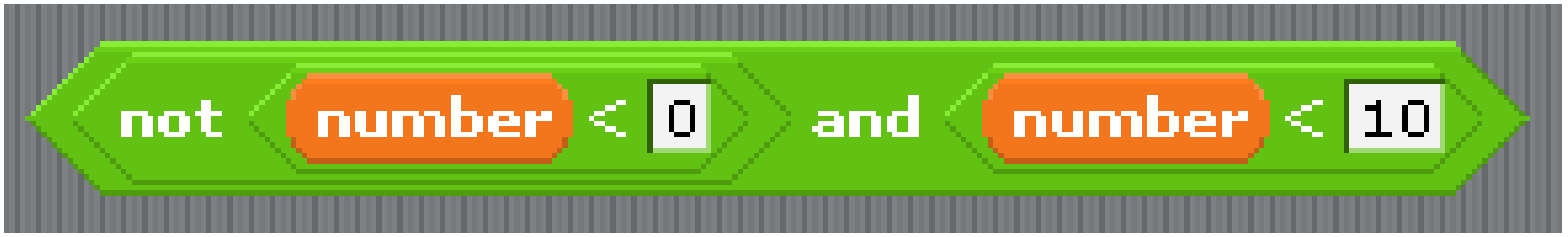
Boolean Operators

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.

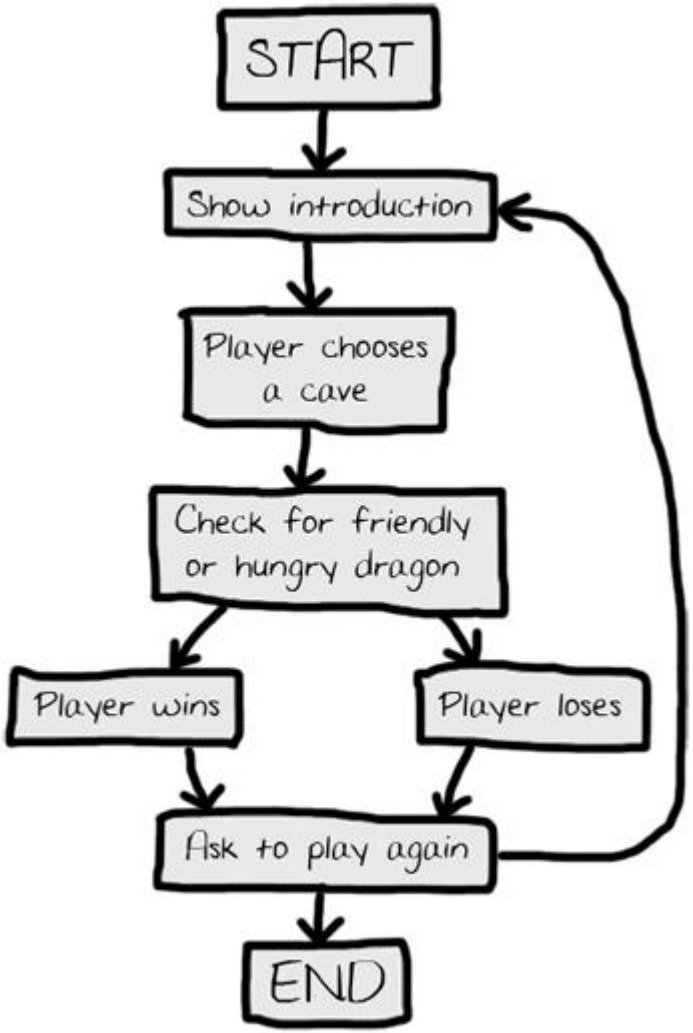
```
while (not number  < 1)  and number < 10:
```



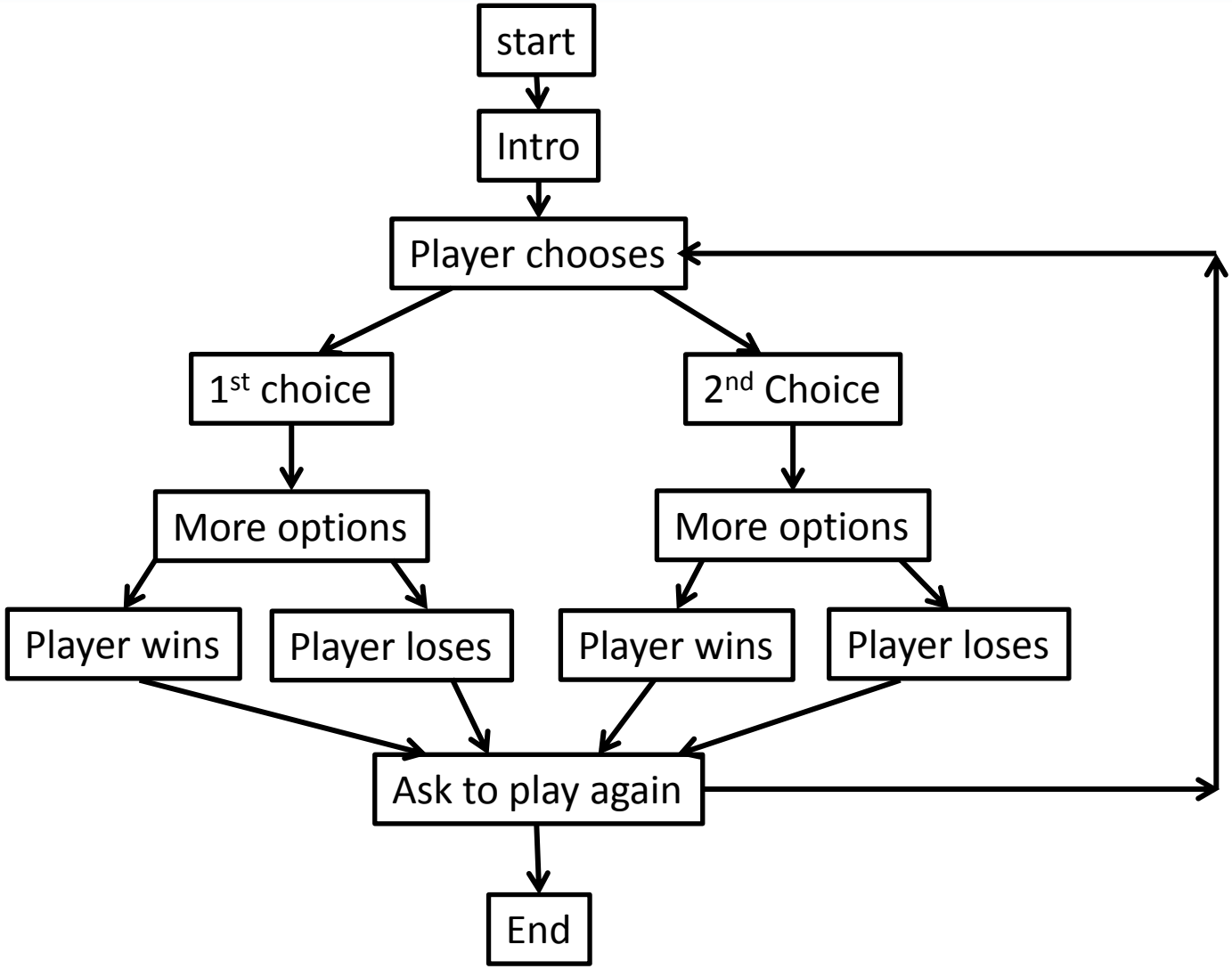
Boolean Operators

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.

Our Own Game


START
Show introduction
Player chooses a cave
Check for friendly or hungry dragon
Player wins
Player loses
Ask to play again
END

Our Own Game


start
Intro
Player chooses
1st choice
2nd Choice
More options
More options
Player wins
Player loses
Player wins
Player loses
Ask to play again
End