Conditionals, Loops, Repeating things, Repeating things

Why control flow statements?

"Control Flow" => the order in which instructions are executed

Classic Computer Science paper by Edsger Dijkstra: "GO TO considered harmful"

Conditionals: "if" statement example

```
# A simple number guessing game
number = int(raw_input("Guess: "))

if number == 5:
    print "You guessed my number."
    print "My number was 5."
print "The end."
```

Conditionals: "if" statement example

```
number = int(raw_input("Guess: "))

# The condition (note the colon after)
if number == 5:
    print "You guessed my number."
    print "My number was 5."
print "The end."
```

Conditionals: "if" statement example

```
number = int(raw_input("Guess: "))

if number == 5:
    # Both of these print statements
    # happen if the condition is True
    print "You guessed my number."
    print "My number was 5."
# Happens regardless of the condition
print "The end."
```

An aside on Python indentation

In Python, whitespace is meaningful. Anything in a code block that needs to be executed together needs to be indented the same amount.

Python PEP8 Style Guide says:

"Always use 4 spaces for indenting."

Breaking down the condition

```
if condition:
    print "Only if condition is True"
```

- "condition" is an expression (code that represents some value)
- "condition" is first converted to a bool type
- If True, the branch is taken
- If False, the branch is skipped
- The entire indented "block" of code is either taken or skipped as a whole.

A few operators

Definition: an operator "operates" on some number of expressions and returns a value. These operators all return True or False.

Equality operator is ==

    e.g. print 4==3 # prints False

Negation operator is not

    e.g. print not False # prints True

Comparison operators are <, <=, >, >=

    e.g. print 5 < 10 # prints True

Arithmetic operators

Addition operator (+): 4 + 5 == 9

Subtraction operator (-): 3 - 4 == -1

Multiplication operator (*): 5 * 12 == 60

Modulo (remainder) operator (%): 14 % 3 == 2

Division operator (/): 10 / 2 == 5

*** Warning ***

3 / 2 == 1 (integer division)

3 / 2.0 == 1.5 (floating point division)

Guess the value! Don't cheat!

```
 1. 4 == 5
 2. not 4 == 5
 3. 1 + 2 < 3
 4. bool(100)
 5. bool(-1)
 6. bool(0)
 7. bool(2.0)
 8. bool(0.0)
 9. bool("abc")
10. bool("")
11. not 0
```

Conditionals: "if" statement example

```
number = int(raw_input("Guess: "))

if number == 5:
    print "You guessed my number."
    print "My number was 5."
if not number == 5:
    print "You didn't guess right!"
print "The end."
```

Conditionals: "if" statement example

```
number = int(raw_input("Guess: "))

if number == 5:
    print "You guessed my number."
    print "My number was 5."
else:
    print "You didn't guess right!"
print "The end."
```

Multiple conditions

```
x = 3
if x == 4:
    print "Only if x == 4"
elif x < 5:
    print "Only if x < 5"
elif x == 3:
    print "Only if x == 3"
else:
    print "No conditions met"
```

Nested conditionals

```
x = 15
if not x < 10:
    if x < 15:
        print "Statement 1"
    else:
        print "Statement 2"
    print "Statement 3"
else:
    if x == 15:
        print "Statement 4"
```

Guessing game: multiple chances

```
number = int(raw_input("Guess: "))
if number == 5:
    print "You guessed my number."
    print "My number was 5."
else:
    print "Wrong!"
```

Guessing game: multiple chances

```
number = int(raw_input("Guess: "))
if number == 5:
    print "You guessed my number."
    print "My number was 5."
else:
    print "Wrong!"
number = int(raw_input("Guess: "))
if number == 5:
    print "You guessed my number."
    print "My number was 5."
else:
    print "Wrong!"
number = int(raw_input("Guess: "))
if number == 5:
    print "You guessed my number."
    print "My number was 5."
else:
    print "Wrong!"
```

One of the most important programmer rules:

"Don't Repeat Yourself"

Loops! The "while" keyword

A general while loop:

```
while condition:
     print "Repeat while condition is True"
```

Compare with a conditional:

```
if condition:
    print "Do this once if condition is True"
```

Beware the infinite loop

```
while True:
    print "Help, I'm stuck in an infinite loop!"
```

Loops! The "for" keyword

```
for x in [3, 2, 1]:
    print x
print "Blastoff!"
```

Loops! The "for" keyword

```
for x in [3, 2, 1]:
    print x


x = 3
print x
x = 2
print x
x = 1
print x
```

Dissecting the for loop: fruit salad

```
Keywords: "for" "in"
Loop variable: "x"
Expression to iterate over: [3, 2, 1]
Don't forget the colon!

for x in [3, 2, 1]:
    print x
```

Every for loop has these four parts!

Loops! Using range()

range(num) is a function that takes an int parameter and returns a list of ints from 0..n-1

e.g. range(5) returns the list [0, 1, 2, 3, 4]

```
# A quick example:
for value in range(10):
     print value + 1
```

Taking a break

A break statement aborts the loop

```
for x in range(100000):
    if x >= 2:
        break
    print x
```

Taking multiple breaks

A break statement only aborts the nearest loop.

```
for y in range(3):
    for x in range(100000):
        if x >= 2:
            break
        print x
    print "Outer loop!"
```

Continuing to take a break

A continue statement "continues" to the next loop iteration without execution any more code in the loop.

```
for x in range(10):
    if x % 2 == 1:
        continue
    print x
```

Continuing to take multiple breaks

A continue statement also only continues the "nearest" for loop.

```
while True:
    for x in range(10):
        if x % 2 == 1:
            continue
        print x
    break
```