DS2000
Fall 2021
Handout: While Loops

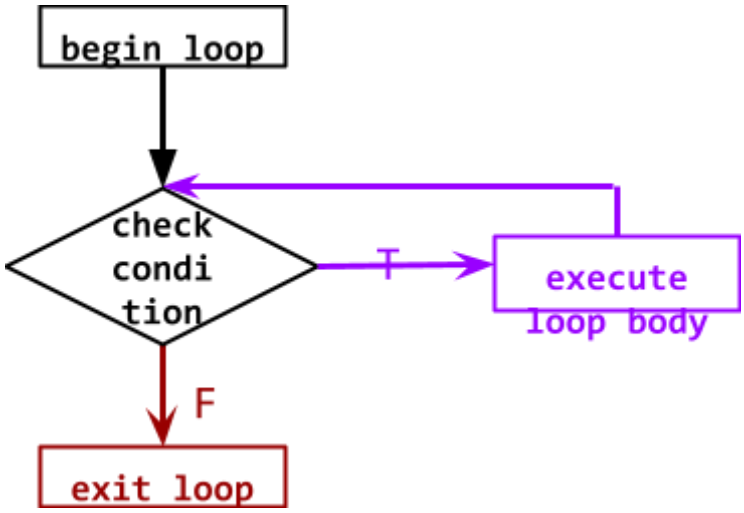
All programming languages must be able to:

1. remember things,
2. repeat things,
3. communicate, and
4. make decisions.

Loops are all about #2: repeating things. We'll learn two kinds of loops in DS2000, while loops and for loops.

While loops are our focus today, and they are "condition-controlled": we use a while loop when we want to repeat a few lines of code over and over and over and we might not know exactly how many times it'll happen.

While loops have the following structure:



The very first thing we do when the loop begins is check the condition; if True, execute the loop body, which is every statement within a **`code block`**. If False, the flow of program control picks up right outside the loop body.

Note that, if the condition is false the first time we check it, we *never* execute the loop body. Just because you see a loop in a piece of code doesn't guarantee that it executes multiple times, or even that it executes one time.

We definitely want to hit "exit loop" at some point, or we'll just keep repeating the loop body forever. FOREVER. Computers don't get tired. In order to exit the loop, one of two things must happen:

The condition becomes False, or
We encounter a break statement.

Example - The condition becomes False

We need to translate our ideas in English into something Python can understand. When planning out a piece of code, we might say "I want to repeat my code block **until** *x* equals 2." In Python that becomes "**While** *x* is not equal to 2, repeat my code block."

Here's how that might look:

```
x = 0
while x < 2:
    print("DUCK...")
    x += 1
print("GOOSE!")
```

Line 4 modifies the value of *x*, so that the condition can change from True to False.

Example - We encounter a break statement

Another way to exit a loop is with a **break** statement. A break statement says, *ignore the rest of this code block and exit the loop*. It's different than exiting when a while condition becomes False, because we could skip out right in the middle of a code block.

We often use this structure to read from a file, especially if we don't know how many lines of text/data are in it. When planning out a piece of code, we might say "I want to read this file **until** I reach the end." In Python that becomes "**While** there is content to read, keep reading."

Here's how I would read a file of integers, with one integer per line, until I hit the end:

```
total = 0
with open("file.txt", "r") as infile:
    while True:
        num = int(infile.readline())
        if not num:
            break
        total += num
```

In this case, the while loop condition never becomes False. Instead, we look for a condition where we want to exit the loop and use the Python reserve word **break** to skip the last line of the code block and exit the loop.