**Flags**

A ***flag*** is a ***Boolean*** ***variable*** that is used to track a ***state***.

A flag in computer science serves the same function as a flag in real life: when an event happens, the flag is either raised or lowered. In real life the event might be:

* you received mail
* the race is over
* you surrender

The point of a flag is to signal an event that may have happened a while ago. For example, your mail flag tells you that you have mail. You don’t have to sit all day and wait to spot the mailman. If he delivers the mail, he will set the flag. If the flag is set, you must have mail. If it isn’t, you don’t have mail.

In programming a flag is a variable that ‘remembers’ a state. A ***state*** might be:

mail/no mail

dead/alive

found/not found

left/right

running/not running

A flag is ***set*** when it is made True:

playerAlive = True # notice the capital T

A flag is ***reset*** when it is made False:

mailReceived = False # notice the capital F

Games use flags everywhere. For example, suppose the UP key is used to make a player jump. We want the key to “trigger” the event (the player is jumping). Once the button is pressed the player jumps. But “jumping” is a process that will take time to play out, and we don’t expect the user to continue to press the key. All the key press does is signals that we are in the jumping phase:

if key == UP\_Button:

playerjump = True # player will start the jump

If the key is pressed, the flag is set. Once the flag is set (True) a separate section of code takes care of the actual jumping part. We will see this later in the course when we get into game programming.

Here is a program that uses a flag to determine if a person entered a password correctly:

success = False

for i in range(3):

password = str(input("Please enter a password: "))

if password == 'hello':

success = True

break

else:

if i<2: print ("Sorry, try again")

print()

if success == True:

print ("Welcome!")

else:

print ("I'm sorry, you are not welcome. Please leave.")

In this example, the flag is named success. Notice:

* it was reset at the start (success = False). This is because the user has not successfully entered the password - yet.
* it was set when the user entered the correct password

When the loop is done, there are two possible reasons:

1. The user got the password right
2. The user took three tries and was unsuccessful.

That is where the flag comes in.

* if the flag was set (to true), the user got it right.
* if the flag was cleared (to false) the user did not get the password.

**Programming with Flags**

There are usually three stages (or more) in the life cycle of a flag. These are:

1. **The flag is set (or reset).** This is done at the start of the block. For example, we might set playerAlive = True.
2. **An event resets (or sets) the flag.** For example, we might write:

If time == 0: playerAlive = False # you ran out of time

This part is usually inside of a loop.

1. **The flag is checked.** Somewhere later in the code, we check the flag and make an action:

If playerAlive == True: showScore() # show that player’s score

**Exercises** *Incorporate a flag into each exercise.*

1. Write a program that asks the user for ten numbers. If any of the numbers is even, have the program write “You picked at least one even number” at the end of your program. If there were no even numbers, output “You picked no even numbers”.
2. Modify #1, by asking for one and only one even number. If the user enters two or more even numbers (or none) let them know. [hint: this will require 2 flags]

1. Write a program that asks for 4 consecutive numbers (ex. 5, 6, 7,8 ). If the numbers are not consecutive, inform the user. If they are consecutive, thank the user.
2. A useful function is a **toggle**. A toggle changes value back and forth given the same input.

Ask the user for a number. Put this request in an infinite loop. If they enter a “5”, have the program output “yes”. The next time they enter “5”, have the output be “no”. The output will continue to toggle back and forth with each entry of the value 5. If the user enters a “0”, have the program quit.

1. Banking Machine Part II

Combine two of your previous assignments: your bank machine (4.1) and your login (6.3).

* The log in should allow you 3 attempts to get the password right. If the user gets it wrong after 3 tries, the program ends.
* Once you are in, the bank machine allows you to withdraw money (no other functions!).
* When the user has finished, allow the user to withdraw money again. Give the user an option to exit the program.
* When the user “exits” the program, go back to the login screen and start all over again.
* The program never ends.

This program will require 3 loops (1 for and 2 while loops) and at least one flag. At least one of your while loops will be infinite.