Cat and Mouse Programming Tutorial

Waverly Roeger
waverlyroeger@gmail.com

Derek Jones
derekjones@asu.edu

Introduction to Programming Concepts

You can use https://www.onlinegdb.com/online_python_compiler to try out everything in this handout.

Comments

"One of the most important things to learn is how to write a line of code that does absolutely nothing."—Waverly

Comments are lines of code that are ignored by the computer and allow the programmer to write notes for readers. In python a "#" before text means that it won't be executed.

```
#This is an example of a comment
```

Print Statements

So now on to code that actually does something.

The print statement allows the program to print something out to the user.

```
print("This is an example of a print statement")
```

Then when this line is executed it will print:

```
This is an example of a print statement
```

Notice how what is printed is enclosed in parentheses and quotation marks. The quotation marks signify that there is text.

Variables

Variables are one of the most important tools a programmer has. A variable is a way to store and recall values. A variable can hold numbers, words, sentences and many other things.

```
myVariable = 0
```

The above code creates a variable, "myVariable", and gives it a value of 0.

If you were to plug the variable into a print statement, it would print its value, which is zero. Notice how there are no quotes around "myVariable" because it is not text, but rather a variable. Variables can have almost any name you want, so try to give them names that help you remember what they store.

```
myVariable = "Hello, World!"
print(myVariable)
```

This will print:

```
Hello, World!
```

You can also update variables. Here is an example of incrementing "myNumber" by 1.

```
myNumber = 1
print("My number is", myNumber)
myNumber = myNumber + 1
print("My number is now", myNumber)
```

Notice you can print text and variables in the same line by listing them with commas in the same print statement. This will print:

```
My number is 1
My number is now 2
```

Variables are one of the most important tools for any programmer to use.

Input

If we want to get input from the user, we can use the "input" function. The input function displays a prompt and then stores what the user inputs to a variable.

```
myWord = input("Please enter a word: ")
print("The word you entered was", myWord)
```

This will print(if the user inputs "Camel" for example):

```
Please enter a word: Camel
The word you entered was Camel
```

Logical Operators

The next building blocks are the logical operators, like your typical operators (+, -, *, /) logical operators take in two values and produce a result. Logical Operators are special in that they can only return True or False.

The logical operators we will cover are == (equals), > (greater than), < (less than). Here are some examples:

```
2 == 2 # True, 2 and 2 are equal
3 > 1 # True, 3 is greater than 1
4 < 7 # True, 4 is less than 7
0 == 1 # False, 0 and 1 are not equal
2 > 5 # False, 2 is not greater than 5
4 < 2 # False, 4 is not less than 2
```

So if you were to feed these into print statements:

```
print(2 == 2)
print(4 < 7)
print(2 > 5)
```

The program would print:

```
True
True
False
```

If Statements

The if statement checks an expression that uses a logical operator and then does something if it is True. For examples:

```
if 4 > 2:
  print("4 is greater than 2")
if 2 > 4:
  print("2 is greater than 4")
```

This will print:

```
4 is greater than 2
```

This is the case because "4 > 2" is true while "2 > 4" is false

While Loops

A while loop is like a special kind of if statement where the contents of the loop will keep running as long as the expression is true. In order to avoid infinite loops (where the condition is always true), be sure to update your variables inside your while loop. Check out the last line of this next code snippet for an example!

```
apples = 1
oranges = 5
while apples < oranges:
  print("We now have", apples, "apples")
  apples = apples + 1
```

This will print:

```
We now have 1 apples
We now have 2 apples
We now have 3 apples
We now have 4 apples
```

Notice how it never says there are 5 apples because when there are 5 apples, apples is no longer less than oranges.

Let's start coding!

Choose a Starting Location for the Cat

First we are going to create 2 variables named "catX" and "catY" to store the location of the cat. In this example, we set the cat's initial position to (0,0).

```
catX = 0                                                    <--
catY = 0                                                    <--
```

Hide the Mouse

Next, we want to hide the mouse somewhere—and in a similar way to the cat this is done with 2 variables: "mouseX" and "mouseY". You can choose any 2 numbers for the mouse but for this example we will use (3,3). Our code should now look like this:

```
catX = 0
catY = 0

mouseX = 3                                                  <--
mouseY = 3                                                  <--
```

Getting Player Input for Movement

At the beginning of the game we want to tell the player where the cat currently is and then take input to find out where to move the cat. You can print text to the screen to tell the player where the cat is with the print statement. Then you can use a variable to store their input. In this case we will name our variable "direction" and store the input in it.

```
print("The cat is currently at", catX, ",", catY)
direction = input("Which direction do you want to go? ")
```

Our code should now look like:

```
catX = 0
catY = 0

mouseX = 3
mouseY = 3

print("The cat is currently at", catX, ",", catY)}          <--
direction = input("Which direction do you want to go? ")    <--
```

Moving the Cat

So now we have a variable "direction" that contains the direction the player wants to move—either "l" for left, "r" for right, "d" for down, or "u" for up. For example. if the player inputs "l", then we should move the cat to the left. To move the cat to the left we need to decrease "catX" by 1. This can be accomplished with an if statement. (Don't forget to indent for the line inside of the if statement!)

```
if direction == "l":
  catX = catX - 1
```

Now, just do the same thing for all 4 directions, like so:

```
if direction == "l":
  catX = catX - 1
if direction == "r":
  catX = catX + 1
if direction == "d":
  catY = catY - 1
if direction == "u":
  catY = catY + 1
```

Our program should now look like:

```
catX = 0
catY = 0

mouseX = 3
mouseY = 3

print("The cat is currently at", catX, ",", catY)
direction = input("Which direction do you want to go? ")

if direction == "l":                                          <--
   catX = catX - 1                                            <--
if direction == "r":                                          <--
   catX = catX + 1                                            <--
if direction == "d":                                          <--
   catY = catY - 1                                            <--
if direction == "u":                                          <--
   catY = catY + 1                                            <--
```

Create the Gameplay Loop

We want the game to run until the cat finds the mouse—until the location of the cat and the location of the mouse are the same. To do this we are going to use a while loop which will loop until "catX" is equal to"mouseX" and "catY" is equal to "mouseY". In the context of a while loop, it will loop while the cat is not in the same position as the mouse. This can be done with the following line:

```
while not (catX == mouseX and catY == mouseY):
```

Our code should now look like this: (Don't forget to indent the contents of the loop!)

```
catX = 0
catY = 0

mouseX = 3
mouseY = 3

while not(catX == mouseX and catY == mouseY):                <--
   print("The cat is currently at", catX, ",", catY)
   direction = input("Which direction do you want to go? ")

   if direction == "l":
      catX = catX - 1
   if direction == "r":
      catX = catX + 1
   if direction == "d":
      catY = catY - 1
   if direction == "u":
      catY = catY + 1
```

Let the Player Know that they Won!

Lastly, we need some way to let the player know that they won. This can be done using a print statement after the while loop, such that it will only occur once the cat has found the mouse. (Make sure this line is NOT indented.)

```
print("You found the Mouse at", mouseX, ",", mouseY)
```

And now, our finished program should look like:

```
catX = 0
catY = 0

mouseX = 3
mouseY = 3

while not(mouseX == catX and mouseY == catY):
   print("The cat is currently at", catX, ",", catY)
   direction = input("Which direction do you want to go? ")

   if direction == "l":
      catX = catX - 1
   if direction == "r":
      catX = catX + 1
   if direction == "d":
      catY = catY - 1
   if direction == "u":
      catY = catY + 1

print("You found the Mouse at", mouseX, ",", mouseY)        <--
```

Go ahead, give it a play! Try changing the "mouseX" and "mouseY" values and letting a friend try to find the mouse.
If anything went wrong and you can't figure it out, a working version can be found at https://onlinegdb.com/ByxP-6s6Q

Challenges

Now that you have finished the introduction, here is a list of things you can try with the code:

- Add a secret coordinate, such that if the cat goes there, it prints a message. (Try to add an additional if statement inside the while loop that checks the cat's location.)

- What if we wanted the mouse's starting position to be random, so that even the author does not know where the mouse could be? This can be done with a python function called "random"!

  For starters, we have to add the following code to the top of our file so that we can access the "random" function.

  ```
  import random
  ```

  Then, where we declare the the location of the mouse we can change the number with a random call, like so:

  ```
  mouseX = random.randint(0,3);
  mouseY = random.randint(0,3);
  ```

  This will make it so that the mouse starts somewhere before 0 and 3 at random in both the x and y.

  A working example can be found at https://onlinegdb.com/SJgUpnipQ

- What if we wanted to limit the cat's movement so that it can't move too far away? We can append the if statements so that if the cat is already at the border, the cat cant move any further in that direction

  ```
  if direction == "l" and catX > 0:
  ```

  This is how we would make it so the cat can't go any farther left than 0—can you try the same for the other directions?

  A working example can be found at https://onlinegdb.com/rkXY9no67

- Now finding a mouse is fun but what if we wanted to let the player know if they are getting closer or farther? This can be done using the distance formula and a variable.

  The distance formula is: $distance = \sqrt{(b_x - a_x)^2 + (b_y - a_y)^2}$

  Like with random, we have to add the following code to the top of our file so that we can access the "sqrt" and "pow" functions.

  ```
  import math
  ```

Second, we need to get the initial distance from the cat to the mouse, this should be done before the while loop. In code this looks like:

```
prev_distance = math.sqrt(math.pow(catX - mouseX, 2) +
   math.pow(catY - mouseY, 2))
```

Then, inside of the while loop we need to find the new distance and see if it is greater than or less than the old distance. Then lastly, we need to set the previous distance to the new distance. The following code accomplishes this:

```
new_distance = math.sqrt(math.pow(catX - mouseX, 2) +
   math.pow(catY - mouseY, 2))

if new_distance > prev_distance:
   print("Colder")
if new_distance < prev_distance:
   print("Hotter")

prev_distance = new_distance
```

A working example can be found at https://onlinegdb.com/SyiTk6jam

- An example with the last 3 listed challenges can be found at https://onlinegdb.com/BkEkWasTQ
- Experiment and try to add your own changes!