ICS3U Case Study: Digit Match

Now that we are able to generate random values and to make decisions, we can implement a simple game in which the user attempts to guess a two-digit number (10-99) randomly selected by the computer. The player scores one point for each digit that is in the same position as the target value. For example, if the target is 72 and the user enters 42, the user scores a point for one match (2).

The first thing to do is `import` the `random` module, so that we can generate a random value between 10 and 99. Once we have done this, we prompt the user to enter their own two-digit value.

```
import random
target = random.randint(10, 99)
print("I'm thinking of a two-digit number.")
print("Each digit that matches one of mine earns a point.")
user_num = int(input("Enter a two-digit positive number: "))
```

We can check whether the value falls between 10 and 99 by linking these two conditions using `and`, like in the code below. If both of these conditions are `True`, then the `if` block will be evaluated with code that we will write later. If the user enters a negative value, the `elif` block will be executed instead. Finally, if all of the previous conditions are `False`, then the user will have entered either a positive values less than 10, or a positive value greater than 99, which is outside of the accepted range. The `else` statement will catch these cases.

```
if user_num >= 10 and user_num <= 99:
    # CODE TO COMPARE TARGET AND USER'S NUMBER
elif user_num < 0:
    print("That's not a positive number.")
else:
    print("That's not two digits.")
```

If the user-entered value is between 10 and 99, then we need to do some analysis. First, we let the user know what the secret number is, and **initialize** a variable that will record how many digits are correct. This code, and all that follows, will replace the comment `# CODE TO COMPARE ...` above.

```
print("The number is ", target, ".", sep="")
correct = 0
```

Next, we need to check both the first and last digits of the number to see if there is a match. It is possible that both digits match, only one digit (first or last) matches, or that no digits match. Obtaining the last digit can be done by checking the remainder after the number is divided by 10. If they do, we add 1 to `correct` to indicate a match.

```
if user_num % 10 == target % 10:
    print("The last digits match.")
    correct += 1
```

Similarly, we can obtain the first digit of the number by obtaining its quotient when divided by 10.

```
if user_num // 10 == target // 10:
    print("The first digits match.")
    correct += 1
```

If there was no match (signified by a value of 0 for `correct`), we display a message to the user. In all cases (two matches, one match, or no matches), we display the number of matching digits.

```
if correct == 0:
    print("Nothing matches.")
print("You scored", correct, "points.")
```

The complete program is below. You should try playing the game several times to verify that it handles all cases with two matches, one match, and no matches.

```
import random
target = random.randint(10, 99)
print("I'm thinking of a two-digit number.")
print("Each digit that matches one of mine earns a point.")
user_num = int(input("Enter a two-digit positive number: "))
if user_num >= 10 and user_num <= 99:
    print("The number is ", target, ".", sep="")
    correct = 0
    if user_num % 10 == target % 10:
        print("The last digits match.")
        correct += 1
    if user_num // 10 == target // 10:
        print("The first digits match.")
        correct += 1
    if correct == 0:
        print("Nothing matches.")
    print("You scored", correct, "points.")
elif user_num < 0:
    print("That's not a positive number.")
else:
    print("That's not two digits.")
```

It may be difficult to verify all cases due to the random nature of the program. Here are some ways in which you might handle this:

- Temporarily overwrite the value of `target` after it has been randomly generated by assigning a known value to it. This will allow you to specify the exact value you want to test.
- Insert a `print` statement with the value of `target` shortly after it is generated. Once testing is complete, this line can be deleted or commented out. This preserves randomness but, since values are random, it may require multiple runs before all cases can be verified.
- Use the debugger to monitor the value of `target`. This does not require the addition or modification of any code. We will discuss how to use the debugger soon.

Exercises

- Modify the program so that it uses a three-digit number instead.
- Using your three-digit program, a player “wins” if two or more digits match and “loses” otherwise. Modify the code so that it announces whether a player wins or loses.
- Using the original two-digit program, modify the code so that a digit matches if it is in any position. For example, if the target number is 53 and the user enters 37, then there is one match (3). Watch out for duplicates: there is only one match for 55 and 58.