Syntax Guide

Indentation allows us to convey the program's structure. Essentially it tells you how the program is running and when statements are executed. For example we have the following two pieces of code:

```
i = 0
while i <= 5:
    i = i + 1
    print(i)
```

```
i = 0
while i <= 5:
    i = i + 1
print(i)
```

Both are loops that increment the value of i until it reaches 5. In the first loop we see the print statement underneath the statement that increases the value of i by 1. In the second block it is outside the while loop (we know this because of how it's indented. In the first block the print statement will print the value of i each time it's incremented because it is inside the while loop. The second block of code will only print it once because that print statement is outside the while loop.

Assignment

In programming the equal sign ("=") is an assignment operator. It does not represent equality. Instead equality is represented by ("=="). For the code below, = means "set to"
The following code:

```
puppies = "great"
```

means:
Assign the variable puppies the value "great". The next line of code prints the value of puppies.

```
print(puppies)
```

Will print

```
"Great"
```

Print Statement

```
print(statement)
```

Prints input to the user. Examples include:

statement = "hello"
print(statement)
OUTPUT: "hello"

print("statement")
OUTPUT: "statement"

print(5)
OUTPUT: 5

print(“5”):
OUTPUT: “5”

six = 6
print(six)
OUTPUT: 6

Counting

Counting for programmers starts at 0

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Arithmetic Operators

In the code below, `=` means “set to” as above, `+` means add `-` means subtract, and `*` means multiply, while == tests whether two things are equivalent.

Modular Arithmetic

Modulo (abbreviated to `mod`) finds the remainder when one number is divided by another
For example, the remainder when 15 is divided by 2 is 1 because 15 divided by 2 is 7 with a remainder of 1.

We write this operation in the following way: `15 mod 2 = 1.`
Here are a few other examples of mod:

15 mod 3 = 0
14 mod 3 = 2
5 mod 8 = 5
2 mod 3 = 2
3 mod 5 = 3
18 mod 3 = 0

The `for` Statement

`for`

This loop is a specialized construct for iterating a specific number of times, often called a "counting" loop. For example (no pun intended):

for i from 1 to 5:
    print(i)
OUTPUT: 1, 2, 3, 4
Note: the for loop ONLY runs through 1,2,3,4 and when i = 5 it considers the loop complete.

You can also use for loops to go through each

Conditionals and Comparison Operators

In the pseudocode for this quiz:

`>=` means "Greater than **or equal to**" so `5 >= 5` is a true statement, as is `6 >= 5`, but `4>=5` is NOT a true statement

`<=` means "Less than **or equal to**" so `5 <= 5` is a true statement, as is `4 <= 5`, but `6<=5` is NOT a true statement

`==` means "check if these two things are the same"
So:

```
puppies = "great"
if puppies == "great":
      print("I love puppies")
```

Will print "I love puppies" because `puppies` is currently set to the value "great" and "great" is the same as "great"

```
puppies = "terrible"
if puppies == "great":
      print("I love puppies")
```

Will NOT print "I love puppies" because puppies is set to "great" and NOT "terrible"

`else if` means "if the condition above wasn't met, check if this condition is met" and `else` means "if none of the ifs or else-ifs had their conditions meant, do this."
So:

```
if 5 < 4:
      print ("first if")
else if 3 < 4:
      print ("else if")
else:
      print("else")
```

Will print "else if" because 5 is NOT less than 4, but 3 IS less than 4.
Whereas:

```
if 5 > 4:
      print ("first if")
else if 3 < 4:
      print ("else if")
else:
      print("else")
```

Will print "`first if`" because 5 is greater than 4. It will never check the else if (even though it's also true in this case), or the else, because the first if was correct.
And:

```
if 5 < 4:
      print ("first if")
else if 3 > 4:
      print ("else if")
else:
      print("else")
```

Will print "`else`" because 5 is NOT less than 4 AND 3 is NOT greater than 4.
This differs from:

```
if 5 > 4:
      print ("first if")
if 3 < 4:
      print ("second if")
else:
      print("else")
```

Which will print both "`first if`" AND "`second if`." Without the "else" in front of the second if, the computer will check the second if even if the first is true.