GERMANNA
ACADEMIC CENTER
FOR EXCELLENCE

Coding: “For” and “While” Loops Handout

A common problem that programming students encounter is knowing when to use a *for loop* versus a *while loop*. This handout explains the differences between each loop and indicates when they should be applied. Because Python and Java programming languages vary in format, this handout includes examples in both languages.

Use the following links to navigate to specific sections:

For Loops

For loops are used when there is a definite number of times that a block of code will be executed. They are commonly used to move through and access items in a container. In the examples below, the program is set to iterate 10 times beginning with the default value 0.

Python Example:

Python Console

```
for x in range(10):
  print(x)
```

```
0 1 2 3 4 5 6 7 8 9
```

In Python, the function *range* is often used with *for loops* when cycling through a set of numbers. The *range* function requires an ending value, and it includes optional parameters for the starting value and step value. The step value is the amount by which the loop will increase. The starting value has a default index of 0, and the step value has a default index of 1.

In the example below, the step value, 2, is added to the starting value,0, until it reaches the ending value, 10.

Python Example:

Python Console

```
for x in range(0,10,2):
   print(x)
```

```
0 2 4 6 8
```

In Python, *for loops* are used to go through each item in a list. A variable within the *for loop* represents the current item from the list during each iteration.

Python Example:

```
Python Console
colors = ["blue","green","red","orange"]
for x in colors:⬅
   print(x, end = " ")
blue green red orange
```

In Java, *for loops* have a different format but are used in the same ways. When creating a *for loop* in Java, there are three key components: the initial value, a conditional statement, and the increment or decrement value.

Java Example:

```
Java Console
for (int i = 0; i < 10; i++) ⬅ {
System.out.print(i + " "); }
0 1 2 3 4 5 6 7 8 9
```

Java Example:

```
Java Console
String [] colors = {"Blue", "Green", "Yellow"};
for (int i = 0; i < colors.length; i++) {
  System.out.print(colors[i] + " "); }
Blue Green Yellow
```

In this example, the variable "i" is used to access the elements in the "colors" array of strings. Initially, "i" is equal to 0, which corresponds to the first element, "Blue". Therefore, "colors[i]" evaluates to "colors[0]" and accesses "Blue" to output in the first iteration of the loop.

Incrementing vs. Decrementing

In Java, the “++” and “--” operators increase or decrease a value by one, respectively. When the symbol is before the variable, the change will be evaluated before the remainder of the line of code. If the operator is after the variable, the change will occur after the rest of the line has been executed.

Java Example:

Java Console

```
int decrease = 5;
System.out.print(decrease--);
```

```
5
```

Java Example:

Java Console

```
int increase = 5;
System.out.print(++increase);
```

```
6
```

The symbols “++ “and “--” are absent from Python since it does not increment or decrement in the same way.

While Loop

Unlike *for loops*, *while loops* can iterate for an indefinite number of times. They are controlled by a condition that must be true for the loop to execute and must be false for the loop to end. Since *while loops* do not have an established end, it is possible to create an infinite loop.

Python Example:

Python Console

```
user_string = input("Enter a phrase:")
while user_string!= "":
  print("You entered: ", user_string)
  user_string = input("Type a new phrase or press ‘enter':")
print("Done.")
```

Input

```
It is a fun day
```

Output

```
You entered: It is a fun day
Type a new phrase or press ‘enter’:
Done.
```

In the example, the user enters a string of characters that become the assigned value of user string. The *while loop* is triggered, printing the original input, and then prompting the user for another string. The *while loop* will compare user string to an empty string until they are equal, at which point the loop breaks.

In this Java example, the *while loop* is controlled by a comparison statement that must remain true for the loop to continue. Inside the *while loop,* the variable “x” is being incremented by 1 until “x” is no longer less than 3.

Java Example:

Java Console

```
int x = 0;
while (x < 3){
  System.out.print(x + " ");
  x++; }
```

```
0 1 2
```

Java has a variation of the *while* loop called a *do while* loop. In this structure, the body of the loop precedes the condition, so it is always executed once before the condition is checked.

Java Example:

Java Console

```
int x = 0;
do {
  System.out.println("Print this.");
  x++; }
while (x < 2);
```

```
Print this.
Print this.
```

Nested Loops

Loops inside other loops are called nested loops. Most often, loops of the same type are used together, but nested loops can also consist of a *for loop* and *while loop*. They are useful when handling complex algorithms that require different outcomes for multiple items.

Python Example:

Python Console

```
first_names = ["Tyler", "Jake"]
last_names = ["Peterson", "Jackson"]
for firstN in first_names:
   for lastN in last_names:
      print(first,last)
```

```
Tyler Peterson
Tyler Jackson
Jake Peterson
Jake Jackson
```

In the example above, the first iteration of the outer loop assigns the “firstN” variable with the value at index 0, “Tyler,” located in the “first_names” list. Then, the program moves to the first iteration of the inner loop and similarly assigns the “lastN” variable with the value at index 0, “Peterson,” in the “last_names” list.

The body of the inner loop prints the values together. In the second iteration of the inner loop, “firstN” remains assigned to the initial index, “Tyler,” while “lastN” is reassigned to the next item in the list, “last_name[1]” or “Jackson.” Once all elements in the container of the inner loop have been accessed, the program returns to the outer loop, reassigning “firstN” with “first_names[1]” or “Jake” and repeating the cycle of assigning “lastN” and printing the values.

Java Example:

Java Console

```
for (int i = 1; i <= 3; i++){
  for (int j = 1; j <= 4; j++){
    System.out.print(i * j + "\t"); }
System.out.println(); }
```

```
1 2 3 4
2 4 6 8
3 6 9 12
```

In the example above, the outer loop initializes “i” with a value of 1 and increments “i” by 1 until “i” is greater than 4. The inner loop follows the same logic, incrementing “j” until it is greater than 3. Therefore, the outer loop runs three times, and the inner loop runs four times per iteration of the outer loop. As a result, the body of the inner loop executes a total of twelve times.

When a new line is printed at the end of every outer loop iteration, the nested loops create a two-dimensional grid with rows and columns corresponding to the number of outer loop iterations and inner loops per outer loop, respectively.

The Academic Center for Excellence (ACE) offers free on-campus and online tutoring appointments for software design. Additional programming resources are available at www.germanna.edu/academic-center-for-excellence. For further assistance with programming concepts, please call ACE at (540) 891-3017 or email ACE@germanna.edu.