CS 315

Homework Assignment 2

Iteration Based on Data Structures in C, Go, Javascript, PHP, Python, and Rust

Name: Funda Tan

ID:21801861

Section: 2

C

1. Iteration statements provided

There are three iteration statements provided in C. These are for, while and do while statements.

For statement

```c
//Iteration statements provided
//For
//String
char string[5] = "apple";
for (int i = 0; i < 5; i++)
  printf ("%d ", string[i]);
printf ("\n");

//Array
int array[5] = { 1, 2, 3, 4, 5 };
for (int i = 0; i < 5; i++)
  printf ("%d ", array[i]);
printf ("\n");
```

Output:

apple

12345

Explanation: In this code segment, we see that using for loop, we can iterate through the array and the string data structure.

While statement

```c
//While
//String
int i = 0;
while (i < 5){
    printf ("%d ", string[i]);
    i++;
}
printf ("\n");

//Array
i = 0;
while (i < 5){
    printf ("%d ", array[i]);
    i++;
}
printf ("\n");
```

Output:

apple

12345

Explanation: In this code segment, we see that using while loop, we can iterate through the array and the string data structure.

```
//Do-While
//String
i = 0;
do{
  printf("%d ", string[i]);
  i++;
} while( i < 5);
printf("\n");

//Array
i = 0;
do{
    printf ("%d ", array[i]);
    i++;
} while (i < 5);
printf ("\n");
```

Output:

apple

12345

Explanation: In this code segment, we see that using do while loop, we can iterate through the array and the string data structure.

2. Data structures suitable for iteration

For for statements, string and array data structures are suitable as we can see in the following example.

```c
//Iteration statements provided
//For
//String
char string[5] = "apple";
for (int i = 0; i < 5; i++)
  printf ("%d ", string[i]);
printf ("\n");

//Array
int array[5] = { 1, 2, 3, 4, 5 };
for (int i = 0; i < 5; i++)
  printf ("%d ", array[i]);
printf ("\n");
```

Output:

apple

12345

Explanation: In this code segment, we see that we can iterate through the array and the string data structure using for loop.

For while statements, string and array data structures are suitable as we can see in the following example.

```c
//While
//String
int i = 0;
while (i < 5){
    printf ("%d ", string[i]);
    i++;
}
printf ("\n");

//Array
i = 0;
while (i < 5){
    printf ("%d ", array[i]);
    i++;
}
printf ("\n");
```

Output:

apple

12345

Explanation: In this code segment, we see that we can iterate through the array and the string data structure using while loop.

For do-while statements, string and array data structures are suitable as we can see in the following example.

```
//Do-While
//String
i = 0;
do{
  printf("%d ", string[i]);
  i++;
} while( i < 5);
printf("\n");

//Array
i = 0;
do{
    printf ("%d ", array[i]);
    i++;
} while (i < 5);
printf ("\n");
```

Output:

apple

12345

Explanation: In this code segment, we see that we can iterate through the array and the string data structure using do-while loop.

3. The way the next item is accessed

For for loops, loop condition is tested before the execution of for loop.

```
//The way that next item accessed
//Loop condition is tested before the execution of for loop
for (int i = 0; i < 5; i++){
  printf ("%d ", string[i]);
  printf("In for loop");
  i = 10;
}
printf ("\n");

for (int i = 0; i < 5; i++){
  printf ("%d ", array[i]);
  printf("In for loop two");
  i = 10;
}
printf ("\n");
```

Output:

a In for loop

1 In for loop two

Explanation: We see that loop executes once, then i becomes 10. After i becomes 10, loop condition is tested before the execution and loop does not execute because loop condition is false.

For while loops, loop condition is tested before the execution of for loop.

```
//Loop condition is tested before the execution of while loop
i = 0;
while (i < 5){
    printf ("%d ", string[i]);
    printf("In while loop");
    i = 10;
}
printf ("\n");

i = 0;
while (i < 5){
    printf ("%d ", array[i]);
    printf("In while loop two");
    i = 10;
}
printf ("\n");
```

Output:

a In while loop

1 In while loop two

Explanation: We see that loop executes once, then i becomes 10. After i becomes 10, loop condition is tested before the execution and loop does not execute because loop condition is false.

For do-while loops, loop condition is tested before the execution of for loop.

```
//Loop condition is tested after the execution of the do while loop
i = 0;
do{
  printf("%d ", string[i]);
  printf("In do while loop");
  i++;
  i = 10;
} while( i < 5);
printf("\n");

i = 0;
do{
  printf("%d ", array[i]);
  printf("In do while loop two");
  i++;
  i = 10;
} while( i < 5);
printf("\n");
```

Output:

a In do while loop

1 In do while loop two

Explanation: We see that loop executes once, then i becomes 10. After i becomes 10, loop condition is tested before the execution and loop does not execute because loop condition is false.

GO (Golang)

1. Iteration statements provided

There are three iteration statements provided in GO. These are for, for without init and post statement and for without conditions statements.

For statement

```
//Iteration statements provided
//For
//String
myString := "apple"
for i := 0; i < 5; i++{
  fmt.Printf("%d ", myString[i])
}
fmt.Println("")

//Array
array := [5]int{ 1, 2, 3, 4, 5 }
for i := 0; i < 5; i++{
  fmt.Printf("%d ", array[i])
}
fmt.Println("")
```

Output:

apple

12345

Explanation: In this code segment, we see that using for loop, we can iterate through the array and the string data structure.

For without init and post statement

```
//For without init and post statements
//String
i := 0
for i < 5 {
   fmt.Printf("%d ", myString[i])
   i++
}

fmt.Println("")

//Array
j := 0
for j < 5{
    fmt.Printf("%d ", array[j])
    j++
}

fmt.Println("")
```

Output:

apple

12345

Explanation: In this code segment, we see that using for loop, we can iterate through the array and the string data structure.

2. Data structures suitable for iteration

For for statements, string and array data structures are suitable as we can see in the following example.

```
//Iteration statements provided
//For
//String
myString := "apple"
for i := 0; i < 5; i++{
  fmt.Printf("%d ", myString[i])
}
fmt.Println("")

//Array
array := [5]int{ 1, 2, 3, 4, 5 }
for i := 0; i < 5; i++{
  fmt.Printf("%d ", array[i])
}
fmt.Println("")
```

Output:

apple

12345

Explanation: For for without init and post statements, string and array data structures are suitable as we can see in the following example.

For for without init and post statements, string and array data structures are suitable as we can see in the following example.

```
//For without init and post statements
//String
i := 0
for i < 5 {
   fmt.Printf("%d ", myString[i])
   i++
}

fmt.Println("")

//Array
j := 0
for j < 5{
    fmt.Printf("%d ", array[j])
    j++
}

fmt.Println("")
```

Output:

apple

12345

Explanation: For for without init and post statements, string and array data structures are suitable as we can see in the following example.

3. The way the next item is accessed

For for loops, loop condition is tested before the execution of for loop.

```
  //The way that next item accessed
  //Loop condition is tested before the execution of the for loop
  for i := 0; i < 5; i++{
    fmt.Printf("%d ", myString[i])
    i = 10
  }
  fmt.Println("")
```

Output:

a

Explanation: We see that loop executes once, then k becomes 10. After k becomes 10, loop condition is tested before the execution and loop does not execute because loop condition is false.

```
  k := 0
  for k < 5{
      fmt.Printf("%d ", array[k])
      k = 10
  }
}
```

Output:

1

Explanation: We see that loop executes once, then k becomes 10. After k becomes 10, loop condition is tested before the execution and loop does not execute because loop condition is false.

Javascript

1. Iteration statements provided

There are five iteration statements provided in Javascript. These are for, for-in, for-of, while and do-while statements.

For statement

```
//Iteration statements provided
//For
//String
   var myString = "apple";
   for(var i = 0; i < 5; i++)
     document.write(myString[i]);
   document.write("\n");

//Array
   var i = 0;
   var myArray = [1, 2, 3, 4, 5];
   for(var i = 0; i < 5; i++){
     document.write(myArray[i]);
   }
   document.write("\n");
```

Output:

apple

12345

Explanation: In this code segment, we see that using for loop, we can iterate through the array and the string data structure.

For-in statement

```
//For-in
//Associative Array
  var myArray2 = {"Name":'Funda', "Surname":'Tan' };
  for(var key in myArray2)
    document.write(myArray2[key] + " ");

//String
  for(const x of myString)
    document.write(x);
  document.write("\n");
```

Output:

Funda Tan

apple

Explanation: In this code segment, we see that using for-in loop, we can iterate through the associative array and the string data structure.

For-of statement

```
//For-of
//Array
  for(const x of myArray)
   document.write(x);
  document.write("\n");

//String
  for(const x of myString)
   document.write(x);
  document.write("\n");
```

Output:

12345

apple

Explanation: In this code segment, we see that using for-of loop, we can iterate through the array and the string data structure.

While statement

```
//While
//String
   i = 0;
   while(i < 5){
     document.write(myString[i]);
     i++;
   }
   document.write("\n");

//Array
  i = 0;
  while (i < 5){
      document.write(myArray[i]);
      i++;
  }
  document.write("\n");
```

Output:

apple

12345

Explanation: In this code segment, we see that using while loop, we can iterate through the array and the string data structure.

Do-while statement

```
//Do-While
//String
  i = 0;
  do{
    document.write(myString[i]);
    i++;
  } while(i < 5);
  document.write("\n");


//Array
  i = 0;
  do{
      document.write(myArray[i]);
      i++;
  } while (i < 5);
  document.write("\n");
```

Output:

apple

12345

Explanation: In this code segment, we see that using do-while loop, we can iterate through the array and the string data structure.

2. Data structures suitable for iteration

For for statements, string and array data structures are suitable as we can see in the following example.

```
//Iteration statements provided
//For
//String
   var myString = "apple";
   for(var i = 0; i < 5; i++)
     document.write(myString[i]);
   document.write("\n");

//Array
   var i = 0;
   var myArray = [1, 2, 3, 4, 5];
   for(var i = 0; i < 5; i++){
     document.write(myArray[i]);
   }
   document.write("\n");
```

Output:

apple

12345

Explanation: In this code segment, we see that we can iterate through the array and the string data structures using for loop.

For for-in statements, associative array and string data structures are suitable as we can see in the following example.

```
//For-in
//Associative Array
  var myArray2 = {"Name":'Funda', "Surname":'Tan' };
  for(var key in myArray2)
    document.write(myArray2[key] + " ");

//String
  for(const x of myString)
    document.write(x);
  document.write("\n");
```

Output:

Funda Tan

apple

Explanation: In this code segment, we see that we can iterate through the associative array and the string data structures using for-in loop.

For for-of statements, array and string data structures are suitable as we can see in the following example.

```
//For-of
//Array
  for(const x of myArray)
   document.write(x);
  document.write("\n");

//String
  for(const x of myString)
   document.write(x);
  document.write("\n");
```

Output:

12345

apple

Explanation: In this code segment, we see that we can iterate through the array and the string data structures using for-of loop.

For while statements, string and array data structures are suitable as we can see in the following example.

```
//While
//String
   i = 0;
   while(i < 5){
     document.write(myString[i]);
     i++;
   }
   document.write("\n");

//Array
  i = 0;
  while (i < 5){
      document.write(myArray[i]);
      i++;
  }
  document.write("\n");
```

Output:

apple

12345

Explanation: In this code segment, we see that we can iterate through the array and the string data structures using while loop.

For do-while statements, string and array data structures are suitable as we can see in the following example.

```
//Do-While
//String
  i = 0;
  do{
    document.write(myString[i]);
    i++;
  } while(i < 5);
  document.write("\n");


//Array
  i = 0;
  do{
      document.write(myArray[i]);
      i++;
  } while (i < 5);
  document.write("\n");
```

Output:

apple

12345

Explanation: In this code segment, we see that we can iterate through the array and the string data structures using do-while loop.

3. The way the next item is accessed

For for loops, loop condition is tested before the execution of for loop.

```
//The way that next item accessed
//For
//Loop condition is tested before the execution of for
loop
//String
   var myString = "apple";
   for(var i = 0; i < 5; i++){
     document.write(myString[i]);
     document.write("In for loop");
     i = 10;
   }
   document.write("\n");
//Array
   var i = 0;
   var myArray = [1, 2, 3, 4, 5];
   for(var i = 0; i < 5; i++){
     document.write(myArray[i]);
     document.write("In for loop two");
     i = 10;
   }
   document.write("\n");
```

Output:

a In for loop

1 In for loop two

Explanation: We see that loop executes once, then i becomes 10. After i becomes 10, loop condition is tested before the execution and loop does not execute because loop condition is false.

For while loops, loop condition is tested before the execution of for loop.

```
//While
//Loop condition is tested before the execution of while
loop
//String
   i = 0;
   while(i < 5){
     document.write(myString[i]);
     i = 10;
     document.write("In while loop");
   }
   document.write("\n");

//Array
  i = 0;
  while (i < 5){
      document.write(myArray[i]);
      i = 10;
      document.write("In while loop two");
  }
  document.write("\n");
```

Output:

a In while loop

1 In while loop two

Explanation: We see that loop executes once, then i becomes 10. After i becomes 10, loop condition is tested before the execution and loop does not execute because loop condition is false.

For do-while loops, loop condition is tested before the execution of for loop.

```
//Do-While
//Loop condition is tested after the execution of while
loop
//String
  i = 0;
  do{
    document.write(myString[i]);
    i = 10;
    document.write("In do-while loop");
  } while(i < 5);
  document.write("\n");


//Array
  i = 0;
  do{
    document.write(myArray[i]);
    document.write("In do-while loop two");
    i++;
  } while (i < 5);
  document.write("\n");
```

Output:

a In do-while loop

1 In for do-while two

Explanation: We see that loop executes once, then i becomes 10. After i becomes 10, loop condition is tested before the execution and loop does not execute because loop condition is false.

PHP

1. Iteration statements provided

There are four iteration statements provided in PHP. These are for, associative array, while and do-while statements.

For statement

```
//For
//String
$string = "apple";
for ($i = 0; $i < 5; $i++)
    echo "$string[$i]";

//Array
$array = array(1, 2, 3, 4, 5);
for ($i = 0; $i < 5; $i++)
    echo "$array[$i]";
```

Output:

apple

12345

Explanation: In this code segment, we see that using for loop, we can iterate through the array and the string data structures.

For-each statement

```
//For-each
//Array
foreach ($array as $x)
    echo "$x";

//Associative Array
$names = array("Ali"=>"0",
"Veli"=>"1");
foreach ($names as $x)
    echo "$x";
```

Output:

12345

01

Explanation: In this code segment, we see that using for-each loop, we can iterate through the array and the associative array data structures.

While statement

```
//While
//String
$i = 0;
while($i < 5){
    echo "$string[$i]";
    $i++;
}

//Array
$i = 0;
while($i < 5){
    echo "$array[$i]";
    $i++;
}
```

Output:

apple

12345

Explanation: In this code segment, we see that using while loop, we can iterate through the array and the string data structures.

Do-while statement

```
//Do-While
//String
$i = 0;
do{
    echo "$string[$i]";
    $i++;
}while($i < 5);

//Array
$i = 0;
do{
    echo "$array[$i]";
    $i++;
}while($i < 5);
```

Output:

apple

12345

Explanation: In this code segment, we see that using do-while loop, we can iterate through the array and the string data structure.

2. Data structures suitable for iteration

For for statements, string and array data structures are suitable as we can see in the following example.

```
//For
//String
$string = "apple";
for ($i = 0; $i < 5; $i++)
    echo "$string[$i]";

//Array
$array = array(1, 2, 3, 4, 5);
for ($i = 0; $i < 5; $i++)
    echo "$array[$i]";
```

Output:

apple

12345

Explanation: In this code segment, we see that using for loop, we can iterate through the array and the string data structures.

For for-each statements, string and array data structures are suitable as we can see in the following example.

```
//For-each
//Array
foreach ($array as $x)
    echo "$x";

//Associative Array
$names = array("Ali"=>"0",
"Veli"=>"1");
foreach ($names as $x)
    echo "$x";
```

Output:

12345

01

Explanation: In this code segment, we see that using for-each loop, we can iterate through the array and the string data structures.

For while statements, string and array data structures are suitable as we can see in the following example.

```
//While
//String
$i = 0;
while($i < 5){
    echo "$string[$i]";
    $i++;
}

//Array
$i = 0;
while($i < 5){
    echo "$array[$i]";
    $i++;
}
```

Output:

apple

12345

Explanation: In this code segment, we see that using while loop, we can iterate through the array and the string data structures.

For do-while statements, string and array data structures are suitable as we can see in the following example.

```
//Do-While
//String
$i = 0;
do{
    echo "$string[$i]";
    $i++;
}while($i < 5);

//Array
$i = 0;
do{
    echo "$array[$i]";
    $i++;
}while($i < 5);
```

Output:

apple

12345

Explanation: In this code segment, we see that using do-while loop, we can iterate through the array and the string data structures.

3. The way the next item is accessed

For for loops, loop condition is tested before the execution of for loop.

```php
//The way that next item
accessed
//For
//Loop condition is tested
before the execution of for loop
//String
$string = "apple";
for ($i = 0; $i < 5; $i++){
    echo "$string[$i]";
    echo "Inside for loop";
    $i = 10;
}

//Array
$array = array(1, 2, 3, 4, 5);
for ($i = 0; $i < 5; $i++){
    echo "$array[$i]";
    echo "Inside for loop two";
    $i = 10;
}
```

Output:

a Inside for loop

1 Inside for loop two

Explanation: We see that loop executes once, then i becomes 10. After i becomes 10, loop condition is tested before the execution and loop does not execute because loop condition is false.

For while loops, loop condition is tested before the execution of for loop.

```php
//While
//Loop condition is tested
before the execution of while
loop
//String
$i = 0;
while($i < 5){
    echo "$string[$i]";
    echo "Inside while loop";
    $i = 10;
}

//Array
$i = 0;
while($i < 5){
    echo "$array[$i]";
    echo "Inside while loop
two";
    $i = 10;
}
```

Output:

a Inside while loop

1 Inside while loop two

Explanation: We see that loop executes once, then i becomes 10. After i becomes 10, loop condition is tested before the execution and loop does not execute because loop condition is false.

For do-while loops, loop condition is tested before the execution of for loop.

```
//Do-While
//Loop condition is tested after the
execution of while loop
//String
$i = 0;
do{
    echo "$string[$i]";
    $i = 10;
    echo "Inside do while loop";
}while($i < 5);

//Array
$i = 0;
do{
    echo "$array[$i]";
    $i = 10;
    echo "Inside do while loop two";
}while($i < 5);
```

Output:

a Inside do while loop

1 Inside do while loop two

Explanation: We see that loop executes once, then i becomes 10. After i becomes 10, loop condition is tested before the execution and loop does not execute because loop condition is false.

Python

1. Iteration statements provided

There are two iteration statements provided in Python. These are for-in and while statements.

For-in statement

```
#For-in
#String
string = "apple"
for value in string:
    print(value)

#List
array1 = [1, 2, 3, 4, 5];
for value in array1:
    print(value)

#Dictionary
names = {'Funda':'Tan', 'Ali':'Veli'}
for name in names:
    print(name)

#Tuple
tuple = ("Apple", "Black", "Blue")
for value in tuple:
  print(value)
```

Output:

apple

12345

Funda Ali

Apple Black Blue

Explanation: In this code segment, we see that using for-in loop, we can iterate through the string, array, dictionary and tuple data structures.

While statement

```
#While
#String
i = 0;
while i < 5:
    print(string[i])
    i = i + 1

#List
i = 0;
while i < 5:
    print(array1[i])
    i = i + 1
```

Output:

apple

12345

Explanation: In this code segment, we see that using while loop, we can iterate through the array and the string data structures.

2. Data structures suitable for iteration

For for-in statements, string, list, dictionary and tuple data structures are suitable as we can see in the following example.

```
#For-in
#String
string = "apple"
for value in string:
    print(value)

#List
array1 = [1, 2, 3, 4, 5];
for value in array1:
    print(value)

#Dictionary
names = {'Funda':'Tan', 'Ali':'Veli'}
for name in names:
    print(name)

#Tuple
tuple = ("Apple", "Black", "Blue")
for value in tuple:
  print(value)
```

Output:

apple

12345

Funda Ali

Apple Black Blue

Explanation: In this code segment, we see that we can iterate through the array and the string data structure using for-in loop.

For while statements, string and list data structures are suitable as we can see in the following example.

```
#While
#String
i = 0;
while i < 5:
    print(string[i])
    i = i + 1

#List
i = 0;
while i < 5:
    print(array1[i])
    i = i + 1
```

Output:

apple

12345

Explanation: In this code segment, we see that we can iterate through the array and the string data structure using while loop.

3. The way the next item is accessed

For while loops, loop condition is tested before the execution of for loop.

```
#The way that next item accessed
#While
#Loop condition is tested before the execution of for loop
i = 0;
while i < 5:
    print(string[i])
    i = 10

#List
i = 0;
while i < 5:
    print(array1[i])
    i = 10
```

Output:

a

1

Explanation: There are two examples in the code sample. First example is with using string data structure and the second one is using array data structure. We see that loop executes once, then i becomes 10. After i becomes 10, loop condition is tested before the execution and loop does not execute because loop condition is false. This is true for both of the examples.

Rust

1. Iteration statements provided

There are three iteration statements provided in Rust. These are loop, for-in and while statements.

Loop statement

```
let mut i = 0;
let array = [0i32; 4];

//Loop
//Array
loop{
    i = i + 1;
    println!("Result: {}", array[i-1]);
    break
}

//Vector
let vector = vec!["a", "b", "c"];
loop {
    i = i + 1;
    println!("Result: {}", vector[i-1]);
    break
}
```

Output:

1

a

Explanation: In this code segment, we see that using loop, we can iterate through the array and the vector data structure.

For-in statement

```
//For-in
//Array
for x in 0..array.len() {
    println!("Result: {}", array[x]);
}

//Vector
for value in vector {
    println!("Result: {}", value);
}
```

Output:

1

2

3

4

a

b

c

Explanation: In this code segment, we see that using for-in loop, we can iterate through the array and the vector data structure.

While statement

```
//While
//Array
i = 0;
while i < 4{
    println!("Result: {}", array[i]);
    i = i + 1;
}

//Vector
i = 0;
let vector = vec!["a", "b", "c"];
while i < 3 {
    println!("Result: {}", vector[i]);
    i = i + 1;
}
```

Output:

1

2

3

4

a

b

c

Explanation: In this code segment, we see that using while loop, we can iterate through the array and the vector data structure.

2. Data structures suitable for iteration

For loop statements, array and vector data structures are suitable as we can see in the following example.

```
let mut i = 0;
let mut array = [0i32; 4];
array[0] = 1;
array[1] = 2;
array[2] = 3;
array[3] = 4;

//Loop
//Array
loop{
    i = i + 1;
    println!("Result: {}", array[i-1]);
    break
}

//Vector
let vector = vec!["a", "b", "c"];
loop {
    i = i + 1;
    println!("Result: {}", vector[i-1]);
    break
}
```

Output:

1

a

Explanation: In this code segment, we see that we can iterate through the array and the vector data structures using loop.

For for-in statements, array and vector data structures are suitable as we can see in the following example.

```
//For-in
//Array
for x in 0..array.len() {
    println!("Result: {}", array[x]);
}

//Vector
for value in vector {
    println!("Result: {}", value);
}
```

Output:

1

2

3

4

a

b

c

Explanation: In this code segment, we see that we can iterate through the array and the string data structure using for-in loop.

For while statements, array and vector data structures are suitable as we can see in the following example.

```
//While
//Array
i = 0;
while i < 4{
    println!("Result: {}", array[i]);
    i = i + 1;
}

//Vector
i = 0;
let vector = vec!["a", "b", "c"];
while i < 3 {
    println!("Result: {}", vector[i]);
    i = i + 1;
}
```

Output:

1

2

3

4

a

b

c

Explanation: In this code segment, we see that we can iterate through the array and the string data structure using while loop.

3. The way the next item is accessed

For loops, loop condition is tested before the execution of for loop.

First example is with array data structure.

```
//The way that next item accessed
//While
//Loop condition is tested before the execution of for loop
i = 0;
while i < 4{
    println!("Result: {}", array[i]);
    println!("Inside while loop");
    i = 10;
}
```

Output:

Result: 1 Inside while loop

Explanation: We see that loop executes once, then i becomes 10. After i becomes 10, loop condition is tested before the execution and loop does not execute because loop condition is false.

Second example is with vector data structure.

```
//Vector
i = 0;
let vector = vec!["a", "b", "c"];
while i < 3 {
    println!("Result: {}", vector[i]);
    println!("Inside while loop two");
    i = 10;
}
```

Output:

Result: 1 Inside while loop two

Explanation: We see that loop executes once, then i becomes 10. After i becomes 10, loop condition is tested before the execution and loop does not execute because loop condition is false.

Evaluation of these languages in terms of readability and writability

In my opinion, C is both writable and readable but it is not writable as Javascript and readable as Python.

Python is readable because it has for-in and while statements, but it is not writable like the other languages. Python can iterate through string, list, dictionary and tuple data structures. So, Python has a great variety of data structures that it can iterate through them.

Also, Rust is similar with Python in terms of iteration statements that are based on data structures.

Javascript is the most writable language in terms of iteration statements that are based on data structures as it has for, for-in, for-of, while and do-while statements. It has the most variety of iteration statements among the six languages. Also, Javascript can iterate through associative arrays.

I think GO is not readable as C, because C has while and do while loops but GO has only the for loop.

Also, PHP is writable too because it has four different loop statements as for, for each, while and do while.

I think the best language in terms of iteration statements based on data structures is Javascript, because it has five different iteration statements based on data structures. It boosts Javascript's writability. Where the other five languages mostly similar to each other, I think Javascript distinguishes itself with variety of its iteration statements.

Learning strategy

First, I used documentations for the languages to get a clear information. They were my main material to find the iteration statements based on data structures. Then, I found the online compilers and start experimenting with these six languages. I was familiar with the syntax of them from the first homework assignment. I experimented iteration statements, data structures and the way that the next item is reached to see the design decisions that these languages are choose to use. Finally, when I finished writing codes for all of the languages, I compared them to find the best language in terms of iteration statements based on data structures. My comparison strategy was looking the variety of the iteration statements and the variety of data structures that can be used. I also compared the languages in terms of writability and readability. The information that I learnt from the lectures also helped me to do this assignment.

The online compilers that I used for this assignment:

C: https://www.onlinegdb.com/online_c_compiler

Go: https://www.tutorialspoint.com/execute_golang_online.php

Javascript: https://www.w3schools.com/js/tryit.asp?filename=tryjs_editor

PHP: https://www.tutorialspoint.com/execute_php_online.php

Python: https://www.programiz.com/python-programming/online-compiler/

Rust: https://play.rust-lang.org/