**Strings**

**Swift**

The strings used in swift are of two types, variable type and constant type.

//For a constant type

let world=”A better place.”

Here world is a string instance which has some value ie enclosed within the quotation marks.

The keyword let, makes *world* a constant. The value when once assigned to the constant cannot be altered. So if the user tries to change it, the compiler generates an error. Therefore, this means that the instance is not mutable.

//For a variable type

var life=”Live a great one”

Here *life* is a variable of the type string. The contents of this instance can be altered. Also, the compiler is smart enough to infer the type of the variable without having to declare it explicitly.

For example,

var life=”Live a great one”

life+=”!”

Therefore the output generated will be,

Live a great one!

WHY?

->When a user creates a string, the compiler generates a replica of the string. The reason why this happens is because the compiler uses the memory efficiently by creating a replica of the instance of the string, so that the original instance is saved. Which means the operations that the user defines, operate only on the replicas first and then to the original instance.

**GO**

String literals in Go can be created using double quotes i.e “Hi Again” or back ticks `Hi Again`.

WHY?

The double quoted strings can-not contain new lines. So when a string is declared in double quotes and the user wants the string to be in a new line then they have to explicitly use special escape sequences like \n (replaced with a new line) and \t ( replaced with a tab character).

The operations the user can perform on the string can be acquiring the length of the string, accessing an individual character and concatenation of two strings.

// For example,

package main

import “fmt”

func main()

{

fmt.println(len(“Hi Again”))

fmt.println(len(“Hi Again”[1]))

fmt.println(len(“Hi”+ “ Again”))

}

Here,

* The space is also considered as a character. Therefore, the first set of instruction will return 8 as the string’s length.
* Since strings are indexed starting at 0, the first element is accessed i.e the letter “i”. But when this statement is executed, the user gets 105 as the value. This is because the character is represented by a byte. ( which is an integer value)
* Concatenation symbol is +. The go compiler infers what to do based on the type of arguments. The compiler assumes that you mean concatenation and not addition, as strings can-not be added.

**Python**

A string in python is a sequence of characters. These are similar to lists, but the only difference is that, these strings are immutable. The individual parts of the string cannot be changed unlike lists.

//For example

a=’Google’

a[2]

* ‘o’

But if the user gives:

a[2]=’i’

A type error is generated saying that the object doesn’t support item assignment.

The indexing in strings start from 0. And the length and concatenation is same for Python as it was in Go. But list methods like *append, delete* and *pop* are not available for strings. If a string needs to be appended then the user should create a new string. The operations that can be performed on the string are *capitalize, find* and i*ndex*.

**Characters**

**Swift**

The characters that are in strings are of the Character type. This is used to represent Unicode characters. These collection of characters, form a string instance.

//For example;

let world=”A better place”

var life=”Live”

life+=”..”

for c: Character in life.characters

{

Print(“ ‘\(c)’ “)

}

This loop iterates through every character in *life*. The user accesses the property of the character of the string *life*. A property is the way a type holds on to data. The properties in swift are accessed via a *dot syntax(.)*.

The print() prints a line break after logging its content.

i.e

‘l’

‘i’

‘v’

‘e’

‘.’

‘.’

**Python**

Python does not support character type, that is why they are treated as strings of length one.

The user can access the character as follows:

Sen1=’Monday’

Sen2=”Blues”

Print “sen1[2]: “,sen1[2]

Print “sen2[1:3]: “,sen2[1:3]

When the above code is executed,

* Sen1[2]: n
* Sen2[1:3]: lue

The indexing method is used to access the character. As the strings are immutable, individual character cannot be changed and assigned to a value.

**Go**

;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;

**Loops**

SWIFT

For loops hold good for iterations that are known and easy to derive, whereas while loops are suited for tasks that execute repeatedly until the condition is met.

1. For-in loop

var aint: Int =0

for i in 1…10

{

aint+=1

aint

print(aint)

}

Here a variable *aint* is declared that is of type int which is initialized be equal to 0. When the user declares the for loop, the iterator *i* that represents the current iteration. This iterator iterates through the range specified by the user, in this case 1 to 10. The incrementing code written in the {} braces is executed for each iteration.

1. While loop

This loop executes as long as the condition is true. A while loop for the for loop above can be expressed as,

Var i=1

While i<11{

Aint+=1

Print(aint)

i+=1

}

This while loop initializes an incrementer(var i=1), evaluates a condition (i<11), executes code if the condition is valid(aint+=1,print(aint),increment i) and then returns to the top of the while loop to determine whether the loop should continue iterating.

*i* is declared as a variable because in the loop the value of *i* must change.

1. If-else loop

var apple: int 10

var basket: string

if apple<10

{

Basket=”\(apple) in the basket.”

}

Else

{

Basket=”\(apple) is not in the basket”

}

Print(basket)

The variable apple is declared as an instance of Int type and the value is assigned. The user then declares basket as the string type. The value is now uninitialized. If the *if statement* evaluates to true, then the basket value updates to the value mentioned in {} and if it is false the else statement value is updated.

**Go**

1. For loop

The for statement in GO looks something like this,

Package main

Import “fmt”

Func main(){

i:=1

for i<=10

{

Fmt.println(i)

I=i+1

}

}

Now this can also be written as,

Func main()

{

For i:=1,i<=10;i++

{

Fmt.println(i)

}

}

The loop works the same way as it does in Swift. Only the declaration part of the loop is different because the statements inside the loop run as blocks, only if the condition is true else the program exits.

\*While loops are inferred as for loops in go. Therefore this PL has no explicit while loop.

1. If else loop:

If i%2==0

{

Fmt.println(“number is even”)

}

Else{

Fmt.println(“number is odd”)

}

The *if* statement is similar to for, but the only difference is that if a false value is inferred in the program, the program doesn’t exit, but instead executes the else block of the program.

**Python**

1. The syntax of a for loop is;

For item in list:

Statements

// The statements must be at the same indentation level as they form the body of the loop. Item-variable name, list-list.

For example:

For i in [1, 2, 3, 4, 5]:

Print I,

* 1 2 3 4 5

The comma is used after print to print the output on the same line.

1. While loop

The syntax of the while loop is:

While condition:

Statements

The condition is evaluated first and if it is true the body is executed and this repeats until the condition evaluates to False.

//For example

A=0

While a<10:

Print a,

A=a+2

* 0 2 4 6 8

// the body of a loop is never executed if the condition evaluates to false for the first time.

**Swift**

**Arrays**

An array is a collection of homogenous values in an ordered list. An empty array is initialized as follows,

Var sum=[int]()

//sum is of type int with 0 items. “.count” is used to find the count in an array. If no data type is declared explicitly then the compiler will count the value to be of type double, this is because the output value of the program is optimized.

Var apple=array(repeating:5, count:5)

* 5 5 5 5 5

\* Creating a new array by adding two arrays

Var basket = array(repeating: 3.0, count 2)

Var basket = apple + basket

-> var apple is inferred as type double and equals [5.0,5.0,5.0,5.0,5.0,3.0,3.0]

\* String type arrays

-> var chocolate: string = [“cadbury”, “five-star”,”munch”]

(The variable chocolate stores string type arrays)

\*Accessing the array elements on a whole:

→Print(“chocolate with me is (chocolate.count) ”)

using the (\_: append) notation,an array can be added to an array

>>chocolate.append = “perk”

—>the array now has 4 elements

Or

Simply by using (+=) addition assignment operator

>> chocolate += [“eclairs”,”gems”]

—> chocolate now contains 6 elements

Retrieving an array value

Var get = chocolate[1]

—> the value of get is equal to “five-star”

Insert at index type,

>> chocolate[1] = “safari”

//the array used in the new programming language, is specified or obtained from Swift, so only the array construct of swift is specified.

Python

Python doesnt have arrays it uses lists. Every element in the list is separated by a comma between square brackets.

All the elements in the list are of the same type.

//for example

chocolate=[1,2,3,4,5]

a list is declared with the elements in the square bracket. These elements can be accessed using the indices.

//for example

chocolate[1]

→ output is 2

lists can also be concatenated by using the plus operator.

//for example

chocolates+[6,7,8]

→ output is 1,2,3,4,5,6,7,8

new items can be added at the end of the list by using append() method.

\*lists are mutable as their value can be changed using indices unlike strings.

Len() is used to get the length of the list.

//for example

len(chocolates) = 8

Requirement Analysis:

The language which we are designing has the following functions and attributes;

Name-SEED

Purpose- For educating the young minds about computers and initializing the seed for curiosity and excitement in them.

This language is the basic for future programming languages the student will learn. Developing his/her interest in coding from this language.

Function: The student can perform basic arithmetic and logical operations on integers and strings. This language will also include geometrical shapes, and various functions which will deal with calculating the area and perimeter of the polygon.

Key requirements

* Readability:

1. In terms of user: Since the primary users of SEED are students, we have designed the syntax and declarations of various constructs in such a way that it appears quite easy for the user to comprehend each instruction or line of code written.
2. In terms of compiler: While compiling the instructions we will make sure that our compiler is smart enough to deal with the instructions and perform the lexical and syntax analysis. Also ensuring the speed of execution.

* Data types:

1. Integer
2. String
3. Decimal
4. Value- boolean
5. Shape- name of polygon

* Syntax:

1. Variable type definition-

Variable <variable name> of <data type> is <variable content>

1. To initialize an array-

Variable <variable name> of <data type> is [x,y,..z]

1. To initialize a string:

Variable <variable name> of *string* is “hello”

* For constants initialization:

Use keyword *let*

Let constant\_name= anyValue; (value cannot be NULL)

* To initialize a shape:

shape\_name:shape[dataType ]

Test Case:1

variable myprog of string is “hello world!”

display (myprog)

output: hello world!

Here the variable is declared as type string with the value assigned to it. The display keyword is used for printing the value or a statement.