Práctica 05

DOCENTE	CARRERA	CURSO
MSc. Vicente Enrique Machaca Arceda	Escuela Profesional de Ingeniería de Software	Fundamentos de Lenguajes de Programación

PRÁCTICA	TEMA	DURACIÓN
05	Ensamblador	3 horas

1. Competencias del curso

- Conocer el desarrollo histórico de los lenguajes de programación y los paradigmas de programación.
- Comprender el papel de los diferentes mecanismos de abstracción en la creación de facilidades definidas por el usuario así como los beneficios de los lenguajes intermedios en el proceso de compilación.

2. Competencias de la práctica

- Programar tareas básicas en lenguaje ensamblador.

3. Equipos y materiales

- Latex
- Conección a internet
- IDE de desarrollo

4. Entregables

- Se debe elaborar un informe en **Latex** donde se responda a cada ejercicio de la Sección 6.
- En el informe se debe agregar un enlace al repositorio Github donde esta el código.
- En el informe se debe agregar el código fuente asi como capturas de pantalla de la ejecución y resultados del mismo.
- Por cada 5 minutos de retraso, el alumno tendrá un punto menos.

5. Datos del estudiante

- Grupo: 03
- Integrante:
  - Jonathan Aguirre Soto

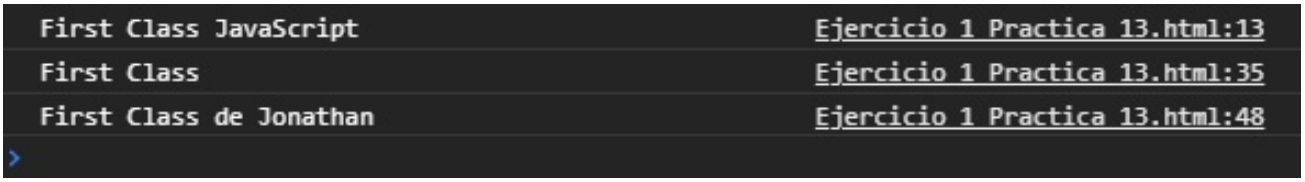
6. Ejercicios

1. Investigue el conecpto de first class en Javascript y muestre una pequeña definición seguida ejemplos. **(2 puntos)**

```
<html>
<body>
<script>
// Definicin :
//Se refiera a cuando las funciones se pueden tratar como otro tipo de dato; es
    decir, asignarles variables y pasarlas como argumentos a otras funciones.

//Primer ejemplo: pasar la funcin como argumento.
 function miFuncion() {
      return "First Class ";
 }

 function miFuncionJava(funcion, java) {
      console.log(funcion() + java);
 }

 miFuncionJava(miFuncion, "JavaScript");

 //Pasamos nuestra funcin miFuncion() como argumento de la funcin
     //miFuncionJava()
   //esto da a entender que tratamos la funcin como un valor

 //Segundo ejemplo; Devolvemos una funcin desde otra funcin.
              function miFuncion2(){
      return function() {
      console.log("First Class");
      }
}

//Ahora, necesitamos invocar la funcin miFuncion() y su funcin interna devuelta.
       //Tenemos dos maneras:

       //Primera manera: usando una variable.

const miFuncion3 = function() {
      return function() {
      console.log("First Class");
      }
}

const miFuncion4 = miFuncion3();
miFuncion4();

       //Segunda manera: usando parntesis doble.

//Usamos parntesis doble ()() para que invoquemos la funcin interna a retornar.

function miFuncion5() {
   return function() {
      console.log("First Class de Jonathan");
   }
```

```
        }
        miFuncion5()();

    </script>
    </body>
    </html>
```

- Ejecución del programa.

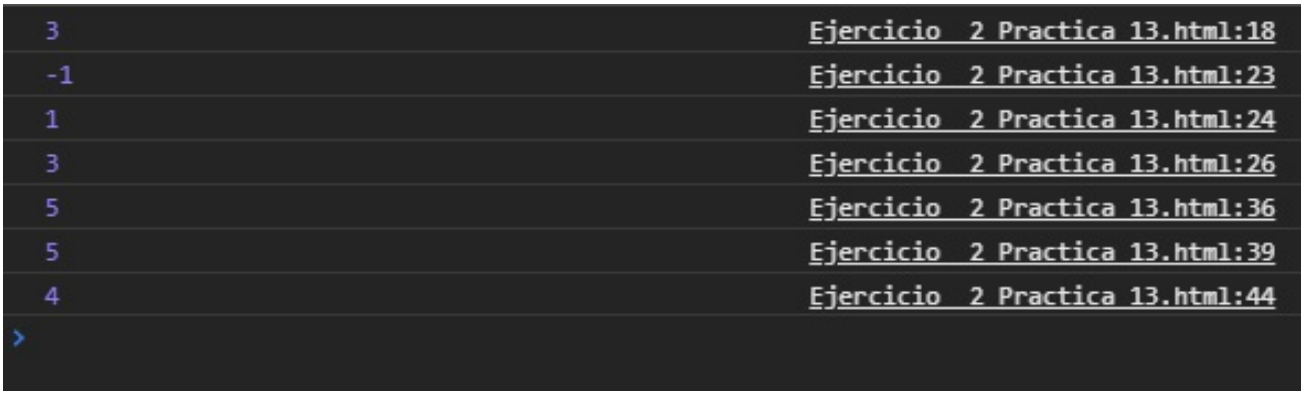


2. Describa la diferencia entre Currying and Partial Application. Incluya ejemplos. **(2 puntos)**

```
<html>
<body>
<script src ="ramda.min.js"> </script>
<script>

//Currying
//Currying es el proceso de descomponer una funcin en una secuencia de funciones
//con un solo argumento. En lugar de tomar todos los argumentos, toma el primero
// y este devolvera una nueva funcion que toma al segundo argumento.

function dot(vect1, vect2){
  return vect1.reduce((acc, ele, i) => acc += ele * vect2[i], 0);
}

const v1 = [1, 3, -5];
const v2 = [4, -2, -1];

console.log( dot(v1, v2));

const curried_dot = R.curry (dot);
const sum_elements = curried_dot([1, 1, 1]);

console.log(sum_elements(v1));
console.log(sum_elements(v2));

console.log(curried_dot(v1, v2));


//Partial application
//Aplicamos parcialmente las funciones a un argumento; es decir; una funcin de
//mltiples  argumentos debido a que las tres funciones que se muestra son
    sintctitamente
//distintas pero semnticamente idnticas. La aplicacin parcial son formas de
//expresar una misma funcin o clculo.

const add = (x, y) => x + y;
console.log(add(2,3));

const addT = y => add(2, y);
console.log(addT(3));

//bind() devuelve una nueva funcin que asegura que 2
//se une al primer argumento.
const add0 = add.bind(null, 2);
console.log(add0(2));

//La diferencia es que currying nos da una secuencia de funciones que en gran
    parte
//siguen siendo la misma que la original; en cambio, la aplicacin parcial produce
//varias funciones para expresar un mismo clculo, la funcin modificada es
    distinta
//a la original pero necesita menos argumentos.

</script>
```

```
</body>
</html>
```

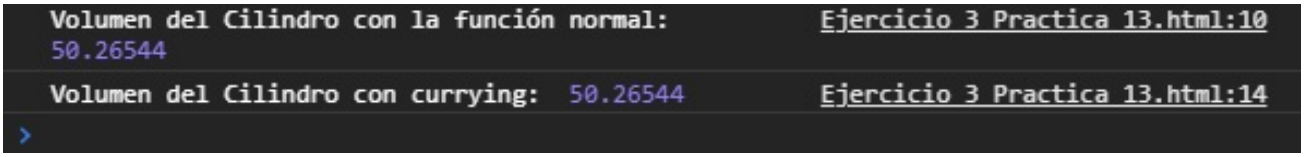
- Ejecución del programa.



3. Implemente una función que calcule el volumen de un cilindro. Incluya la versión normal y una aplicando Currying. **(2 puntos)**

```
<html>
<body>
<script>

// Funcin  Normal

 function volumen_cilindro(radio, altura){
    return (3.14159*(radio*radio)*altura);
 }
 console.log("Volumen del Cilindro con la funcin normal: ", volumen_cilindro(2,
     4));

// Funcin  usando Currying
 const volumen_cilindro_cr = radio => altura => 3.14159*(radio*radio)*altura;
 console.log("Volumen del Cilindro con currying: ", volumen_cilindro_cr(2)(4));

</script>
</body>
</html>
```

- Ejecución del programa.

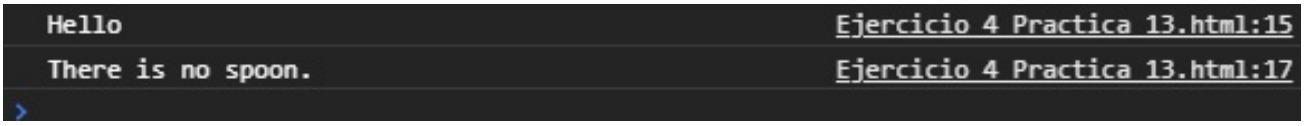


4. Cree una función joinWords que una varios parametros de tipo string. **(3 puntos)**

```
result = joinWords (Hello    ) () ;
console .log ( result ); // Hello
result = joinWords (There    )( is     )( no     )( spoon  .   ) () ;
console .log ( result ); // There is no spoon .
```

```
<html>
<body>
<script>

 const joinWords = string_1 => string_2 =>{
    if(string_2 === undefined){
       return string_1;
    }
    else{
         return joinWords(string_1 + ' ' + string_2);
    }
        };

 result = joinWords ('Hello') ();
        console .log ( result ); // Hello
        result = joinWords ('There')('is')('no')('spoon.') ();
        console .log ( result ); // There is no spoon

</script>
</body>
</html>
```

- Ejecución del programa.

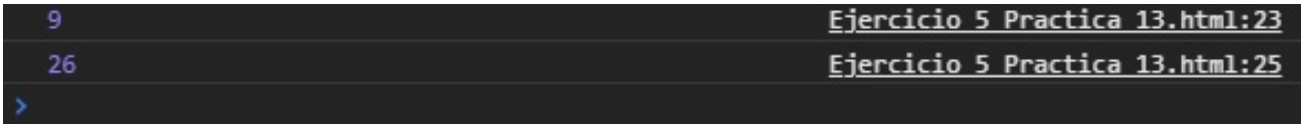


5. Implemente una función delayInvoc que en cada invocación incremente la variable total con el valor enviado como parametro. **(3 puntos)**

```
var total = 0;
var delayInvoc = function (a) {
// your code here
};
delayInvoc (4) (5)
console .log ( total ); //9
delayInvoc (4) (5) (8)
console .log ( total ); // 26
```

```
<html>
<body>
<script>
 var total = 0;

 var delayInvoc = function (a){

    total = total + a;

    return function add (b){

       if(b != undefined){

          total = total + b;
          return add;
       }

       return total ;
    }
 }

 delayInvoc (4) (5)
        console .log ( total ); //9
        delayInvoc (4) (5) (8)
        console .log ( total ); // 26

</script>
</body>
</html>
```

- Ejecución del programa.



6. Implemente una función curry que tome como argumento cualquier función f y retorne la versión curried de f. **(4 puntos)**

```
function abc (a, b, c){
return a+b+c;
}

function curry (f) {
// your code here
}

var curriedAbc = curry ( abc );
console .log ( curriedAbc (2) (3) (4) ); // 9
console .log ( curriedAbc (2 ,3) (4) ); // 9
console .log ( curriedAbc (2) (3 ,4) ); // 9
console .log ( curriedAbc (2 ,3 ,4) ) ; // 9
```

```
<html>
<body>
<script>
 function abc (a, b, c){
    return a+b+c;
 }

 function curry (f) {

    return function curry (){

       const variables = Array.prototype.slice.call(arguments);

       if(variables.length >= f.length){

          return f.apply(null, variables)
       }

       else{
          return curry.bind(null, ...variables)
       }
    }
 }

 var curriedAbc = curry(abc);
 console.log("Funcion Curried");
 console.log(curriedAbc(2)(3)(4)); // 9
 console.log(curriedAbc(2 ,3)(4)); // 9
 console.log(curriedAbc(2) (3 ,4)); // 9
 console.log(curriedAbc(2 ,3 ,4) ) ; // 9

</script>
</body>
</html>
```

- Ejecución del programa.

7. Enlace de Github:
   `https://github.com/Jona2010/Fundamentos-de-Lenguaje-de-la-Programaci-n`

7. Rúbricas

Rúbrica	Cumple	Cumple con obs.	No cumple
Informe: El informe debe estar en Latex, con un formato limpio y facil de leer.	4	2	0
Implementación: Implementa la funcionalidad correcta de cada ejercicio.	16	8	0
Errores ortográficos: Por cada error ortográfico, se le descontara 1 punto.	-	-	-