Tarea 1

Lenguajes de Programación

1. Explica brevemente qué tipos de problemas puedes resolver con cada uno de los siguientes paradigmas y nombra un lenguaje perteneciente a cada uno.
    a. Paradigma Estructurado

    Es de gran ayuda si se desea tener control total de la memoria, utilizarla y liberarla, debido a que se basa en una serie de instrucciones específicas y el mejor ejemplo es *C*.
    b. Paradigma Orientado a Objetos

    Con la herencia es más sencillo mantener algunas características de objetos específicos (incluyendo comportamiento) y además se pueden agregar más características propias; y esto se ve presente en *JAVA*.
    c. Paradigma Funcional

    Es útil en la resolución de cualquier problema matemático sencillo ya que al basarse en funciones y con las instrucciones correctas, se da un resultado certero como lo hace *HASKELL*.
    d. Paradigma Lógico

    Ayuda a resolver problemas de deducción natural, puesto que se basa en reglas y hechos que pueden ayudar a encontrar una conclusión concreta; un ejemplo de esto sería el lenguaje *PROLOG*.

2. Usando el lenguaje de programación Python, dar un ejemplo de cada uno de los siguientes conceptos y justificar. No es necesario que sean ejemplos demasiado elaborados.
    a. Sintaxis

    `3 + 8`

    Simplemente se siguen las convenciones del lenguaje para hacer una simple suma: dos números y un operador.
    b. Semántica

    `x = 70`

A la variable x se le está dando un significado conciso y claro, en este caso el número 70.

c. Convenciones de programación (Idioms)

---

for i in range(0, 10):

Esta forma de declarar un *bucle for* es **exclusiva** de python. No hay ningún cambio en el comportamiento, la diferencia es la forma en la que se declara.

d. Bibliotecas

---

import numpy

---

numpy es una biblioteca que posee métodos preestablecidos que ayuda con el manejo de matrices y vectores así como de funciones matemáticas de alto nivel.

3. Dada la siguiente función, da una firma para la misma indicando el tipo de entrada de los parámetros formales, el tipo de la salida y asígnale un nombre mnemotécnico. Justifica tu respuesta.

```
(define (foo n l)
        (cond
          [(zero? n) l]
          [else (foo (sub1 n) (cdr l))]))
```

```
;; Procedimiento que elimina los n primeros elementos de una lista
;; eliminaNprimeros :: number (listof a) → (listof a)
(define (eliminaNprimeros n l)
        (cond
          [(zero? n) l]
          [else (eliminaNprimeros (sub1 n) (cdr l))]))
```

---

4. Para los siguientes incisos, calcular el resultado de aplicar la función al parámetro real recibido. Mostrar cada paso realizado hasta obtener el resultado final. Da tu propia implementación para ambas funciones.

```
;; Procedimiento que devuelve la reversa de una lista
;; reverse :: (listof a) → (listof a)
(define (reverse lst)
    (cond
        [(empty? lst) l]
        [else (append (reverse (cdr lst)) (list (car lst)))]))

;; Procedimiento que concatena dos listas
;; reverse :: (listof a) (listof a) → (listof a)
(define (append lst1 lst2)
    (cond
        [(empty? lst1) lst2]
        [(empty? lst2) lst1]
        [else (cons (car lst1) (append (cdr lst1) lst2))]))
```

---

a. (reverse '(1 7 2 9))

```
> (reverse '(1 7 2 9))
> (append (reverse '(7 2 9)) (list 1))
> (append (append (reverse '(2 9)) (list 7)) (list 1))
> (append (append (append (reverse '(9)) (list 2)) (list 7)) (list 1))
> (append (append (append (append (reverse '() (list 9))) (list 2)) (list 7)) (list 1))
> (append (append (append (append '() (list 9))) (list 2)) (list 7)) (list 1))
> (append (append (append (append '() '(9))) (list 2)) (list 7)) (list 1))
> (append (append (append '(9) (list 2)) (list 7)) (list 1))
> (append (append (append '(9) '(2)) (list 7)) (list 1))
> (append (append '(9 2) (list 7)) (list 1))
> (append (append '(9 2) '(7)) (list 1))
> (append '(9 2 7) (list 1))
> (append '(9 2 7) '(1))
> '(9 2 7 1)
```

b. (append '(m a n) '(z a n a))

```
> (append '(m a n) '(z a n a))
> (cons '(m) (append '(a n) '(z a n a)))
> (cons '(m) (cons '(a) (append '(n) '(z a n a))))
> (cons '(m) (cons '(a) (cons '(n) (append '() '(z a n a)))))
> (cons '(m) (cons '(a) (cons '(n) '(z a n a))))
> (cons '(m) (cons '(a) '(n z a n a)))
> (cons '(m) '(a n z a n a))
> '(m a n z a n a)
```

c. (reverse (append '(m a n) '(z a n a)))

```
> (reverse (append '(m a n) '(z a n a)))
> (reverse (cons '(m) (append '(a n) '(z a n a))))
> (reverse (cons '(m) (cons '(a) (append '(n) '(z a n a)))))
> (reverse (cons '(m) (cons '(a) (cons '(n) (append '() '(z a n a))))))
> (reverse (cons '(m) (cons '(a) (cons '(n) '(z a n a)))))
> (reverse (cons '(m) (cons '(a) '(n z a n a))))
> (reverse (cons '(m) '(a n z a n a)))
> (reverse '(m a n z a n a))
> (append (reverse '(a n z a n a)) (list m))
> (append (append (reverse '(n z a n a)) (list a)) (list m))
> (append (append (append (reverse '(z a n a)) (list n)) (list a)) (list m))
> (append (append (append (append (reverse '(a n a)) (list z)) (list n)) (list a)) (list m))
> (append (append (append (append (append (reverse '(n a)) (list a)) (list z)) (list n)) (list a)) (list m))
> (append (append (append (append (append (append (reverse '(a)) (list n)) (list a)) (list z)) (list n)) (list a)) (list m))
> (append (append (append (append (append (append (append (reverse '()) (list a)) (list n)) (list a)) (list z)) (list n)) (list a)) (list m))
> (append (append (append (append (append (append (append '() (list a)) (list n)) (list a)) (list z)) (list n)) (list a)) (list m))
> (append (append (append (append (append (append (append '() '(a)) (list n)) (list a)) (list z)) (list n)) (list a)) (list m))
> (append (append (append (append (append (append  '(a) (list n)) (list a)) (list z)) (list n)) (list a)) (list m))
> (append (append (append (append (append (append  '(a) '(n)) (list a)) (list z)) (list n)) (list a)) (list m))
> (append (append (append (append (append '(a n) (list a)) (list z)) (list n)) (list a)) (list m))
> (append (append (append (append (append '(a n) '(a)) (list z)) (list n)) (list a)) (list m))
> (append (append (append (append '(a n a) (list z)) (list n)) (list a)) (list m))
> (append (append (append (append '(a n a) '(z)) (list n)) (list a)) (list m))
> (append (append (append '(a n a z) (list n)) (list a)) (list m))
> (append (append (append '(a n a z) '(n)) (list a)) (list m))
> (append (append '(a n a z n) (list a)) (list m))
> (append (append '(a n a z n) '(a)) (list m))
> (append '(a n a z n a) (list m))
> (append '(a n a z n a) '(m))
> '(a n a z n a m)
```

5. Da una tabla donde expliques las principales diferencias entre un compilador y un intérprete.

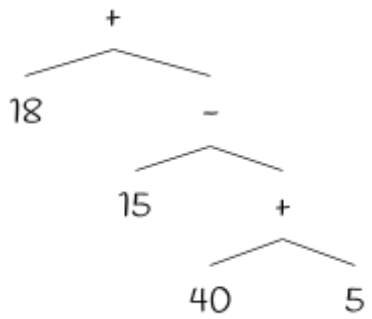
intérprete	compilador
la traducción se realiza durante la ejecución*	la traducción se realiza antes de la ejecución*
la traducción se realiza línea por línea*	todo el código se traduce*
la velocidad de traducción es rápida*	la velocidad de traducción es lenta*
la eficiencia de traducción en baja*	la eficiencia de traducción es alta*
el costo es bajo*	el costo es alto*

* IONOS Digital Guide. (2020). *Compilador e intérprete: definición y diferencias.* Recuperado el 08 de junio de 2021 de https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/compilador-e-interprete/

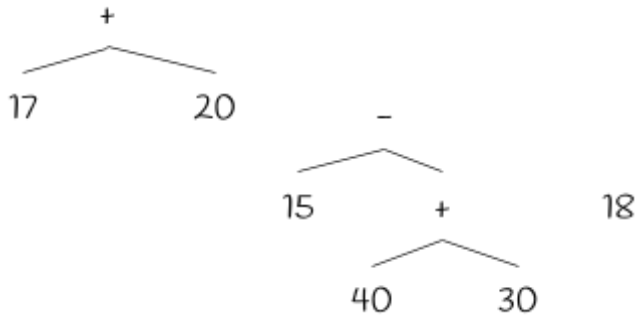
6. Dibuja un mapa mental que muestre las fases de generación código ejecutable, sus principales características y elementos involucrados.

7. Dadas las siguientes expresiones de AE en sintaxis concreta, da su respectiva representación en sintaxis abstracta por medio de los Árboles de Sintaxis Abstracta correspondientes (en forma de lista). En caso de no poder generar el Árbol, justificar.

a. {+ 18 {– 15 {+ 40 5}}}



b. {+ 17 20 {– 15 {+ 40 30 18}}}



Como se puede ver, el ASA no puede ser construido. Si se vuelve a revisar la gramática:

<AE> ::= <num>
| {+ <AE> <AE> }
| {– <AE> <AE> }

es notorio que sólo es posible realizar una operación con dos operando y la expresión dada anteriormente contiene hasta tres.