Administración. Hay cosas que debes saber acerca de este curso.

Quisiera enfatizar los objetivos de este curso. Primero quisiera hablar "**Que serás capas de hacer cuando este curso acabe?** Y luego quisiera hablar de **conceptos** y **herramientas** para el **pensamiento computacional**, que es nuestro principal objetivo en este curso.

Nuestro objetivo (junto con el auxiliar) es **ayudarte a aprender como pensar** como un ingeniero informático o de sistemas. Queremos **que estés listo para los futuros cursos que tomaras en el futuro a lo largo de tu carrera.**

Objetivos.

Objetivos Estratégicos.

1. Queremos que estén **preparados para cursos mas avanzados de computación** como elementos, algoritmos y teoría de grafos, especialmente aquellos estudiantes que no tiene experiencia en programación. Si estas en esta categoría no entres en pánico, este curso es para ti.
2. Queremos que te sientas **confiado en tu habilidad de escribir y leer pequeñas porciones de código.**
3. Para todos los estudiantes, principalmente queremos que e**ntiendan el rol de la computación, que puede y que no puede hacer respecto a resolver problemas técnicos.** En este sentido se darán cuenta que tipo de cosas pueden o no pueden hacer para resolver problemas.
4. Finalmente, para los estudiantes que quieren tener **trabajos de oficina o pasantías**. Queremos que tengan la **confianza en sus habilidades y competencias** para resolver problemas computaciones.

Este curso es principalmente para ayudar a los estudiantes que poca o nada de experiencia en programación. Como consecuencia, asumimos que todos los estudiantes aquí están calificados para este curso: por su puesto todos son estudiantes de la UMSS, tienes que ser bueno para esta aquí!!!!!.

Por otro lado, pedirles a los estudiantes que están "sobre-calificados" para este curso, que ayuden a sus compañeros que tiene menos experiencia.

Objetivos Tácticos
Okay, ahora hablemos que cosas un poco mas técnicas sobre el curso. Que sabrás hacer cuando acabe y apruebes este curso? Que competencias adquirirás?

5. Queremos que seas capas de usar **herramientas básicas para el pensamiento computacional** y **escribir pequeños programas**. Por si acaso, pequeño no significa simple o que menospreciemos lo que seremos capaces de hacer.
6. Queremos que uses **vocabulario** de las herramientas computacionales con el fin de entender programas que escriben otras personas. Esto es importante, no queremos que hagas todo desde cero, en este sentido puedes mirar el código de otra persona, entenderlo y decidir si este funciona bien y puedes agregarle cosas.
7. De tu lado queremos que entiendas fundamentalmente las **capacidades y limitaciones** de las diferentes computaciones. Por ejemplo, hay cosas que son muy difíciles de computar.
8. Principalmente, queremos que seas capaces de tomar la **descripción de un problema y pasarlo a algo computacional (que se pueda computar).**

Ahora debes estar pensando, momento esto se parece mucho a gramática en el colegio. Vamos a aprender que podemos y que no podemos hacer, pero mas importante nosotros vamos a empezar con la habilidad de tomar la descripción de un problema de algún dominio, y pensar como pasarlo a un dominio computacional, de forma tal que tu puedas leer y escribir lo que quieres hacer.

Okay, tranquilo tranquilo, al principio veremos algunos problemas sencillos y con el tiempo terminaras lidiando con problemas mas complejos.

Recomendaciones Generales: Bueno antes de empezar con el curso en si, quisiera decir:

- Este curso **no** es acerca de **memorizar**. No es cuan bien memorizas, de hecho no me gustan las pruebas de memoria. los exámenes probaran to habilidad de pensar.
- La **asistencia es obviamente no obligatoria**. No estamos en colegio. Pero si escoges no asistir a una clase, entonces debes entender que en las clases tendremos un poco de menos paciencia si preguntas cosas que fueron enseñadas en clases pasadas. Esperamos que tengas un comportamiento responsable. De nuestra parte también lo haremos así.
- **No hay textos, ni apuntes de clase.** Se que suena un poco draconiano, pero creo que uno aprende mas tomando notas en la clases.

Entonces, queremos ayudarte a pensar como ingeniero en computación. Es una oración interesante. Que significa pensamiento computacional? Por primero que te quedara de este curso es la noción de resolución de problemas computacionales, esta habilidad se trata de pensar en forma computacional. A diferencia de otros cursos, memorizar no te ayudara. Es mas que todo la noción de las herramientas que utilizaremos.

Formalidades
2 examenes parciales.
2 tareas. Las tareas valen el 20% de cada parcial.
1 examen final. (Quedan eximidos los que aprueben en parciales)
1 instancia.

Etica: principios morales que gobiernan el comportamiento de una persona o grupo de personas.

- cosas que no se debe hacer:
  - Preguntar por tiempo adicional para las tareas o examenes.
  - Reclamar por algo que no entendieron, si antes no preguntar en clase o vía email.
  - Decir que son autores de un pedazo de código que en realidad no lo son.
  - Hacer la tarea a ultimo minuto.
  - Pedir 1 o 2 puntos adicionales, si no han participado en clase.
- Cosas que son bienvenidas a hacer.
  - Buscar inspiración en internet.
  - Hacer tareas con anticipación y preguntar dudas.
  - Pedir ayudas sobre algunas dudas.
  - Asistir a clase.
  - Interactuar con sus compañeros para resolver la tarea.

Empecemos

Pero pensemos: **como definimos pensamiento computacional?** Aunque tal ves lo que deberíamos empezar a pensar es. **Que es computación?**

Pensamiento computacional, **es una extraña oración no es cierto?** Que es computación… parte de la razón por la que pregunto esto para separar el concepto de computadora y computación perse.

Primero subamos un nivel mas arriba. Empecemos con una pregunta mas filosófica: **que es conocimiento?** Veras en dos minutos a que quiero llegar con esto, primero dividamos el conocimiento en dos categorías: declarativo y imperativo.

Conocimiento declarativo.

Okay, volvemos de nuevo que es conocimiento declarativo. Este es un hecho, una afirmación de la verdad. Pongamos un ejemplo:

- La raiz cuadrad de x es y si y solo si y elevado al cuadrado es x, para todo y positivo.

Todos conocemos eso, quiero decir que este es un **afirmación de la verdad, es una definición,** un **aximona**. Eso no ayuda a entender como sacar la raíz cuadrado. Por ejemplo, digamos que x es 2, y queso saber la raíz de 2, tal ves un genio dirá okay es 1,41529 o lo que sea que sea en realidad. Pero aun así no te ayudara a encontrar la raíz.

Si vemos de cerca la ese hecho básicamente es una prueba, y esa prueba no te dirá como sacar la raíz cuadrada.

Conocimiento imperativo.

Por el otro lado, tenemos el **conocimiento imperativo.** El imperativo se la descripción de como deducir cosas. Por ejemplo, déjenme compartir un poco de conocimiento imperativo.

Esta es una pequeña pieza de conocimiento imperativo atribuida a Heron de Alejandria, aunque los babilonios afirmen que sabían eso mas antes.

1. Adivinamos y
2. Pregunto si y*y es cercano a x, si es así lo encontramos.
3. Sino, trato de adivinar un nuevo probando con (y+x)/2.

No se preocupen esto funcionara Heron lo probo. Entonces adivinamos, probamos y repetimos.

Esto señores es una receta, una descripción de un conjunto de pasos. Miren que belleza, es una secuencia especifica de instrucciones en orden. Esto te dice como encontrar la raíz cuadrada. Este es un conocimiento del como. Es un conocimiento imperativo.

Y eso señores es lo que trata computación. Queremos formas de capturar este proceso. Ahora viene una pregunta mas importante aun: “como construimos un proceso mecánico para capturar un conjunto de computaciones?” Lo primero que uno piensa es un pequeño tablero con un montón de circuitos que actualmente podrían realizar una computación en particular.

Fixed-program computers.

Okay, hay que aceptar que eso suena extraño. Pensemos mejor en las primeras computadoras que se llamaban: Fixed-program computers: estas tenia un circuido diseñado para hacer computaciones muy especificas. Por ejemplo, la calculadora, hace computaciones simples sumas, operaciones. Es una computadora que hace operaciones especificas.

Pensemos en otras computadoras de este estilo por ejemplo Atanasoft, en 1941. Una computadora diseñada para dar soluciones a ecuaciones lineales. Algo poderoso si pensamos en la época de los ochentas. Tal ves la mas popular fue la hecha por AlanTuring, llamada la bombe, que fue diseñada para romper códigos. Usados en la segunda guerra mundial. Que obviamente resolvía problemas específicos.

Stored-program computers.

Okay, ahora cambiemos el juego ligeramente. Suponga, que queremos una maquina (o circuito si quiere) que pueda tomar una receta, una descripción de pasos com entrada y luego que la maquina actué como al receta describe. Se re-configurar, que cambien de a cuerdo a la computación que se necesita.

Eso seria asombroso, y adivinen que esa maquina existe. Se llama interprete. Es básicamente el corazón de cualquier computadora.

Ahora hablemos de una computadora que almacena programas. Que significa esto, que uno le puede proveer una secuencia de instrucciones que describen un proceso que queremos que se ejecute.

Dentro de la maquina existe un proceso que permitirá que la secuencia sea ejecutada tal como se describe en la receta.

En realidad es algo bonito de tener, entonces veamos como debería verse de forma mas visual. Dentro de de un stored-program computer, tenemos lo siguiente:

- tenemos una memoria, que esta conectada a dos cosas: unidad de centro, que es llamada **ALU**, aritmética logic unit, y este toma un entrada y saca una salida.
- Por ejemplo tenemos una **secuencia de instrucciones** que están almacenadas ahí. Note que la secuencia de instrucciones en realidad es leída, tratada como datos. Dentro de la memoria
- Esto significa que se puede acceder a ella y se la puede cambiar para construir nuevas piezas de código así como interpretarlas.

También se tiene un **contador de programa** para apuntar a alguna dirección de la memoria, típicamente a la primera instrucción de la secuencia. Todas esas instrucciones son simples, cosas como: tomar dos valores d ella memoria, multiplicarlos en una especie de circuito y luego ponerlos de nuevo en la memoria. Una ves ejecutada una instrucción el contador incrementa en uno y se mueve a la siguiente instrucción.

También existen algunas instrucciones que son pequeñas prueba, que prueban si algo es verdad, si la respuesta es verdad, esta instrucción cambiara el valor del contador del programa y apuntara algún otro lugar de la memoria. Eventualmente tendremos suerte y todo parara de ejecutarse.

Eso es el corazón de una computadora. La noción con la que construimos estas descripciones, nuestras recetas, la secuencias de instrucciones primitivas. Y luego tenemos el control de flujo. El cual se mueve al rededor de la secuencia, ocasionalmente cambiando.

Bueno, entonces que podemos concluir de todo esto, eso seria pensar todo esto como una receta. En realidad un programa es una receta. Una secuencia de instrucciones. Ahora, una cosa que dejamos en el aire es. La receta se construye a través de primitivas, la pregunta seria **que primitivas puedo usar? Cuales son las mas útiles?**

Asumiendo que existen un conjunto de instrucciones primitivas en las cuales se pueden describir todo. Necesitamos saber que cosas puedo construir.

Por ejemplo, separa 6 huevos, revuélvelos hasta que estén en punto crema, luego mezclado con azúcar y agua. En fin una secuencia de instrucciones. Una receta tradicional en realidad esta basada en un pequeño conjunto de primitivas, con las que un buen chef podría cocinar. Lo mismo pasa con programación. Dado un conjunto finito de primitivas, un buen programador puede programa cualquier cosa.

Bueno, entonces ahora la pregunta es, **cuales son las primitivas correctas?** Hay un poco de historia aquí, en 1936, el mismo tipo, Alan Turing, mostró que con **6 primitivas**, cualquier cosa se podría describir como un proceso mecánico, lo que actualmente es algorítmico.

Piensen en eso por un momento. Es una afirmación super increíble. solo 6 primitivas? con solo 6 puedo gobernar el mundo. Una consecuencia de esto, es que uno puede decir que cualquier cosa que hagas en un lenguaje de programación, lo puedes hacer en otro lenguaje.

Asumiendo eso podríamos decir que ningún lenguaje es mejor que otro, aunque la verdad es que no es cierto, existen lenguajes que son mejores para algunas cosas, pero no hay nada que no puedes hacer en C que no puedas hacer en Fortran, esto se llama compatibilidad de Turing.

Entonces la ultima cosa que necesitamos saber antes de hablar de programación real es, que necesitamos describir esas recetas. Bueno, para describir las recetas necesitamos un lenguaje. Necesitamos conocer no solo las primitivas, sino también cosas del lenguaje. Ahora en verdad tenemos cientos de lenguajes de programación.

En verdad existe una gran discusión de que lenguaje seria el mejor para enseñar a programar, no voy a discutir mucho al respecto. Pero en el curso llevaremos al lenguaje JAVA.

Hablemos un poco de lenguajes.

- **high-level o low-level language:** básicamente dice cuan cerca esta a la maquina. Los lenguajes de bajo nivel, que usualmente le llamamos ensamblador, que sus primitivas están al nivel de literalmente mover piezas de datos de una posición de memoria a otra. Un lenguaje de alto nivel, que es mas rico en términos de primitivas, por ejemplo sacar la raíz cuadrada de un numero seria una primitiva fácil de usar, en ves que describir como sacar la raíz cuadrada.
- **General vs targeted language:** donde las primitivas soportan un amplio rango de aplicaciones, o en realidad están orientadas a un conjunto de aplicaciones especificas. Como por ejemplo MATLab que es un lenguaje objetivo.
- **Interpretado vs Compilado:**
  - En un lenguaje interpretado, uno toma lo que se llama código fuente, y el interprete básicamente lo interpreta, la maquina ejecuta el flujo de control de cada instrucción.
  - En un lenguaje compilado, uno tiene un paso intermedio, en el cual uno toma el código fuente, y se ejecuta el compilador, que crea código objeto. Y hace dos cosas, uno es detectar errores y también convierte el código en una secuencia mas eficientes de instrucciones, que al final se ejecutan.

Okay, volvamos al objetivo que es crear recetas. Nuestro objetivo es tomar problemas y romperlos en un monto de pasos computacionales, una secuencia de instrucciones que nos permitan capturar el proceso de solución.

Para eso, necesitamos describir: no solo, que son la primitivas, sino también necesitamos hablar del lenguaje y la interacción con la computadora. Pero antes de hablar de elementos del lenguaje, necesitamos hacer una pequeña distinción. Tal como el lenguaje natural, vamos a separar la sintaxis y la semántica.

Entonces que es sintaxis? Bueno básicamente determina que expresiones son legales en un lenguaje. Por ejemplo, “mi vacaciones mas”. Es una secuencia de palabras en español, pero no están sintácticamente correctas. En verdad no es una oración no tiene ni verbo. Lo mismo que pasa en nuestro lenguaje. Tenemos que describir como poner junto un total de expresiones bien formadas.

Quiero hablar un poco de semántica del lenguaje.

- **Static semantics,** que básicamente dice que programas con significativos. Que expresiones tienen sentido. Aunque sean sintácticamente sean correctas.
- **Full semantics,** lo que es: que significa el programa. Que pasa si lo corro? Cual es el significado de la expresión? Que es lo que pasa? En verdad es lo que quería que pase?

Okay, todo esto parece mucho pre-ambulo. Empecemos con JAVA, pero recuerden el objetivo es hacerles ver lo que trata la computación, por que la necesitamos.

Valores (tipos de datos primitivos)

Tenemos números de diferentes sabores pero por ahora solo nos interesan: enteros y reales.

2
2.51234123
False
True
'a' <— un caracter
"cadena" <— una cadena

También permite operadores básicos y muy conocidos.
+
-
/
*
|| <— o lógico
&& <— y lógico

Como en aritmética podemos combinar estos para formar expresiones matemáticas mas complejas.

2+3*5

Precedencia de operadores

15	Parentesis	()
	Arreglos	[]
	Acceso a miembros	.
14	Post-incremento	++
	Post-decremento	--
13	Pre-incremento	++
	Pre-decremento	--
	Mas	+
	Menos	-
	Negación lógica	!
12	Multiplicación	*
	División	/
	Módulo	%
11	Adición	+
	Sustracción	-
9	Menor que	<
	Menor o igual que	<=
	Mayor que	>
	Mayor o igual que	>=
8	Igual a	==
	No igual a	!=
7	Si binario	&
6	O exclusivo binario	^
5	O inclusivo binario	|
4	Si logico	&&
3	O logico	||
1	Asignacion	=