Tecnologías de la Información II

BSTI2202 (Semestral)
BTTI1002 (Tetramestral)
Notas de enseñanza dirigidas al docente

UNIVERSIDAD TECMILENIO

Herramientas

Para que aproveches al máximo tu experiencia educativa en esta modalidad de certificados, te recomendamos revisar estos **tutoriales**.

Módulo 1

Introducción

Los algoritmos suelen ser soluciones a problemas o procedimientos para lograr un objetivo, independientes de la entidad que pondrá en acción estos procedimientos o soluciones.

En el área de las ciencias de la computación son las computadoras digitales las que terminan desarrollando estas acciones y ejecutando estos programas, por lo tanto, durante el diseño de un algoritmo es importante entender la forma en la que las computadoras digitales entienden, procesan y ejecutan las instrucciones del algoritmo.

Una computadora digital no es tan versátil como un ser humano, aunque hay áreas del conocimiento que buscan dotar a las computadoras de estas capacidades de raciocinio, lo más común es aprovechar las ventajas en velocidad y precisión al momento de realizar operaciones básicas para compensar la falta de razonamiento y deducción.

Notas para el profesor impartidor del tema:

1. Iteraciones

Una iteración es una repetición dentro de un ciclo que debe ejecutarse un número definido de veces.

Una computadora digital no es capaz de razonar, deducir, inferir o generar nuevo conocimiento como lo haría un ser humano, sin embargo, las computadoras digitales son capaces de realizar operaciones lógicas de manera mucho más rápida y precisa que los seres humanos, por lo tanto, los algoritmos computacionales buscan aprovechar esta ventaja descomponiendo problemas complejos en subproblemas mucho más simples, aun cuando estos subproblemas resultan tediosos y repetitivos.

Por ejemplo, si a un humano se le pide calcular el resultado de $4 \times 6 \times 2$, este, dependiendo de su experiencia, puede utilizar su gran memoria y la capacidad de deducir y razonar para resolverlo de la siguiente forma:

Por memoria debe saber que $4 \times 6 = 24$ y el doble de eso es 48.

Mientras que para una computadora no es posible almacenar todas las tablas de multiplicar en la memoria ni tampoco es eficiente buscar el resultado en la misma memoria.

Si se intentara imitar el comportamiento humano, la computadora necesitaría buscar el resultado de 4×6 en su memoria y después buscar el resultado de 24×2. Esta estrategia obligaría a la computadora a tener almacenada cualquier posible multiplicación de números, lo cual es imposible.

Entonces, el programador opta por aprovechar las ventajas de la computadora digital al descomponer la acción de multiplicar en operaciones más simples:

$$4 \times 6 = 4 + 4 + 4 + 4 + 4 + 4 = 24$$

$$24 \times 2 = 24 + 24 = 48$$

Las áreas del conocimiento, como la inteligencia computacional, el aprendizaje automático y el aprendizaje profundo, entre otras, buscan generar procedimientos o metodologías para que una computadora digital sea capaz de imitar el razonamiento humano o de algún otro ente en la naturaleza.

Si bien, este procedimiento requiere de seis sumatorias y aunque para un ser humano esto puede ser tedioso e ineficiente, a una computadora digital le tomaría una fracción de segundo realizarlas. Las computadoras digitales son capaces de hacer operaciones lógicas en milmillonésimas de segundo.

De hecho, la frecuencia de los procesadores, muchas veces mal considerada como la velocidad del procesador, indican el tiempo que a la computadora le toma realizar una operación básica. Por ejemplo, a un procesador de 2 GHz le toma $\frac{1}{2 \times 10^9} = 5 \times 10^{-10}$ segundos realizar una operación básica.

Cabe mencionar que las operaciones básicas de una computadora digital son mucho más simples que una suma, de hecho, la operación de sumar dos números se descompone en varias operaciones aún más sencillas, las cuales se comentarán en el punto número 7.

Entonces, queda claro que los procedimientos repetitivos e iterativos son parte medular de cualquier algoritmo implementado en una computadora digital. La capacidad de entender el problema general y descomponerlo en varias subacciones se adquiere con experiencia y práctica, por lo que resulta importante insistir, con los

aprendedores, en buscar siempre la forma de adquirir nuevos conocimientos que les permitan simplificar sus códigos, por ejemplo, en la actividad “Nivel de práctica: Estén Atentos, *Tengshe*” es posible identificar un patrón en las acciones a realizar.

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Sin embargo, para expresar un procedimiento iterativo que ejecute las mismas acciones que el código mostrado se requieren diferentes combinaciones en las condiciones que desencadena cada acción, lo que implica el uso de condicionales compuestos (tema 6), así que el código mostrado producirá la solución más eficiente que se puede generar hasta el momento, pero después de concluir el tema 6 tendrán una herramienta para hacer más eficientes los códigos y esto sucederá continuamente durante sus vidas profesionales, es decir, la habilidad de utilizar iteraciones se encuentra en constante desarrollo.

Notas para el profesor impartidor del tema:

2. Bucles e iteraciones con condicionales *IF*

El bucle o *loop* es la estructura base para crear un algoritmo iterativo. El bucle se refiere al conjunto de instrucciones necesarias para delimitar los procedimientos a repetir y establecer las condiciones para continuar o detener la repetición o iteración.

Las condiciones de iteración pueden ser tan simples como completar un número perfectamente definido de iteraciones o tan complejas como la combinación de consecuencias dentro del mismo procedimiento.

Por ejemplo, de vuelta a la multiplicación, la condición que nos indica cuántas veces repetir el proceso de sumar es un conteo desde 1 hasta el valor de alguno de los multiplicandos.

Pseudocódigo para resolver la multiplicación de los números enteros a y b

$$a \times b \; a, b \in \mathbb{Z}$$

Define la variable “contador” y asígnale un valor inicial de 1:
contador = 1
Define la variable “resultado” y asígnale un valor inicial de 0:
resultado = 0
Asigna la etiqueta “inicio” al inicio del programa:
inicio:
Realiza la sumatoria del valor almacenado en “resultado” y el valor de a:
resultado = resultado + a
Incrementa en uno el valor de “contador”:
contador = contador + 1
Verifica si el valor del contador es igual que b, si es así, despliega el resultado:
if contador = b
imprime (resultado)
Si no regresa a inicio:
else goto inicio

El algoritmo:

```
contador = 1
resultado = 0
inicio:
resultado = resultado + a
contador = contador + 1
if contador = b:
        imprime (resultado)
else:
        goto inicio
```

Representa un bucle que repetirá el procedimiento (resultado + a) b veces; b es el número de iteraciones del bucle y los condicionales “if, else” corresponden a las condiciones de iteración.

Este procedimiento es el más simple y básico para lograr un bucle que suele conocerse como **iteraciones con condicionales if**, pues se logra generar un bucle a partir de solo condicionales if, else.

Estas estructuras son tan comunes en los programas de cómputo digital que se han generado instrucciones o procedimientos que automatizan o simplifican la implementación de bucles.

2.1 Bucle o ciclo *for* como un condicional if

El primer procedimiento implementado en Python y muchos otros lenguajes para generar bucles es el ciclo for. Dependiendo del lenguaje de programación, la sintaxis, es decir la forma de escribir la instrucción, cambia, aunque todos comparten los elementos básicos de un bucle:

a. El contador de iteraciones.
b. La o las condiciones de iteración.
c. El procedimiento o instrucciones por repetir.

La estructura de un ciclo for en Python es como sigue: se inicia con la instrucción “for”, seguida de la condición de iteración que termina en dos puntos “:”, en una línea o líneas diferentes y con una sangría de un tabulador se colocan las instrucciones a repetir.

for “condición de iteración”:
    procedimiento por repetir 1
    procedimiento por repetir 2
    procedimiento por repetir n

Sangría

La sangría de un tabulador se refiere al espacio entre la columna donde se ubica la instrucción for y las instrucciones a repetir dentro del bucle. El tabulador es un espacio definido entre palabras generado por la tecla Tab del teclado.

Esta separación entre las columnas de los bloques de código que componen a un programa se llama **indentación** que sirve para facilitar la identificación visual de bloques funcionales de código en un programa para el programador o el usuario. Hoy en día, también se utiliza con la misma finalidad por los lenguajes de programación.

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Condición de iteración

La parte de la condición de iteración en el ciclo for permite repetir el procedimiento para cada elemento contenido en un conjunto, por ejemplo, el código:

```
for i in [1, 3, 5, 7, 9]:
   print(i, end=", ")
```

Despliega los números 1, 3, 5, 7 y 9 separados por una coma. Cualquier procedimiento, comando o parte del programa que genere un conjunto de elementos puede ser usado como condición de iteración dentro de un ciclo for, por ejemplo:

```
for i in [Naranja, Coco, Mandarina, Manzana]:
   print(i, end=", ")
```

Enlistará las cuatro frutas separadas por una coma.

Uno de los comandos más utilizados para generar el conjunto de elementos que controlan las iteraciones del ciclo for es “range()”. Esta instrucción genera un conjunto de números a partir de una secuencia numérica, en donde podemos definir el inicio, el fin y el incremento en esta secuencia. Por ejemplo:

```
for i in range(5):
   print(i, end=", ")
```

Despliega la sucesión de números 0, 1, 2, 3, 4. Cuando se utiliza solo un parámetro en la instrucción “range()”, Python interpreta que es un incremento unitario y comienza desde 0. Un segundo parámetro define el inicio de la sucesión:

```
for i in range(-2, 5):
   print(i, end=", ")
```

Despliega -2, -1, 0, 1, 2, 3, 4. Un tercer parámetro define el incremento o paso de la sucesión:

```
for i in range(-2, 5, 2):
   print(i, end=", ")
```

Despliega -2, 0, 2, 4.
Una vez entendido esto, es fácil generar el programa para calcular una multiplicación a partir de sumatorias[2]:

```
For i in range (a):
        resultado = resultado + a
```

Esto genera el resultado de $a \cdot a^1$.

2.2 El ciclo o bucle *while*

Otra estructura que permite generar procedimientos iterativos es el ciclo while, que históricamente también se conoce como “do while” por la sintaxis usada en los primeros lenguajes de programación.

La instrucción “while” permite repetir un procedimiento o serie de instrucciones, **mientras** se cumple o no una condición, por ejemplo, el código:

```
dia = 0

semana = ['Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes', 'Sabado', 'Domingo']
while dia < 7:
  print("Hoy es " + semana[dia])
  dia += 1
```

Desplegara el siguiente mensaje:

```
Hoy es Lunes
Hoy es Martes
Hoy es Miércoles
Hoy es Jueves
Hoy es Viernes
Hoy es Sabado
Hoy es Domingo
```

Si alguien quiere verificar el código de la multiplicación como sumatoria, puede visitar: https://www.programiz.com/python-programming/online-compiler/ y correr el siguiente código:

```
a = 10
resultado = 0
for i in range (a):
        resultado = resultado + a
print(resultado)
```

La instrucción “dia += 1” incrementa en 1 el valor de día. Este ejemplo es útil para recomendar no utilizar caracteres especiales al momento de definir variables; la variable “dia” podría haber sido nombrada “día” de manera más correcta, sin embargo, esto puede dificultar la interpretación de nuestro código por colegas de

otros países, o peor aún, el compilador podría no identificar el símbolo “í” y marcar un error que nos tomaría un poco de tiempo identificar y corregir.

En el caso de cadenas de caracteres, como “Miércoles”, no hay problema, pues es una sucesión de símbolos que el lenguaje solo va a desplegar sin la necesidad de interpretarla.

De vuelta a la instrucción “while”, nos podemos dar cuenta de que la sintaxis es muy similar a la de un ciclo for: instrucción seguida de la condición de iteración terminando en dos puntos.

while dia < 7:

Y el bloque de instrucciones a repetir se coloca a un tabulador de distancia de la instrucción “while”.

2.3 Diferencias entre while y for

Con suficiente imaginación y creatividad es posible obtener el mismo resultado con ambas instrucciones, sin embargo, la instrucción “while” está pensada para definir condiciones más complejas de iteración, pues pueden utilizarse condiciones de mayor que, menor que e igualdad, junto con los operadores lógicos básicos unión, intersección y negación (descritos en el tema 6), lo que da origen a los condicionales compuestos. Mientras que el ciclo for está diseñado para trabajar con listas, vectores, bases de datos, etc., es decir, arreglos de datos o conjuntos de datos.

Como ya se mencionó, es posible generar un procedimiento con cualquiera de las dos instrucciones, alguna de ellas brindará un código más pequeño o eficiente, computacionalmente hablando, y es labor del programador poder identificar esa ventaja operativa en sus implementaciones, aunque no siempre suele ser claro y suele depender mucho de la experiencia y habilidad del programador.

Notas para el profesor impartidor del tema:

3. Anidación

La anidación o *nesting* es una estrategia para simplificar o compactar códigos que requieren de la repetición de repeticiones. Esta estrategia no ofrece ninguna eficiencia computacional al momento de ejecutar las instrucciones, pero sí ofrece eficiencia en líneas de código, lo que permite almacenar el programa en menor cantidad de memoria y consumir menos recursos de cómputo al momento de compilar.

También ofrece una ventaja visual para los programadores con experiencia, lo que facilita la comprensión de los programas para los diferentes miembros de un equipo de trabajo.

La estrategia es muy sencilla, consiste en colocar un bucle (for o while) dentro de otro bucle (for o while) respetando la sintaxis de estos ciclos y estableciendo la jerarquía de los procedimientos a partir de la indentación:

```
Bucle1 condición1:
        Bucle2 condición2:
                Instrucciones por repetir del bucle 2.
        Instrucciones por repetir del bucle 1.
```

Un ejemplo que ilustra las ventajas de la anidación es el algoritmo que despliega las tablas de multiplicar del 1 al 10. Comúnmente haríamos el código que haga la tabla del 1, otro código que haga la tabla del 2 y así hasta la tabla del 10:

```
for j in range(1, 11):
    print(1 * j, end=' ')

for j in range(1, 11):
    print(2 * j, end=' ')
.
.
.
for j in range(1, 11):
    print(10 * j, end=' ')
```

Sin embargo, es evidente que se trata de una repetición de ciclos for, así que, si utilizamos la estrategia de anidación, el código resultante es el que sigue:

```
for i in range(1, 11):
  for j in range(1, 11):
    print(i * j, end=' ')
  print()
```

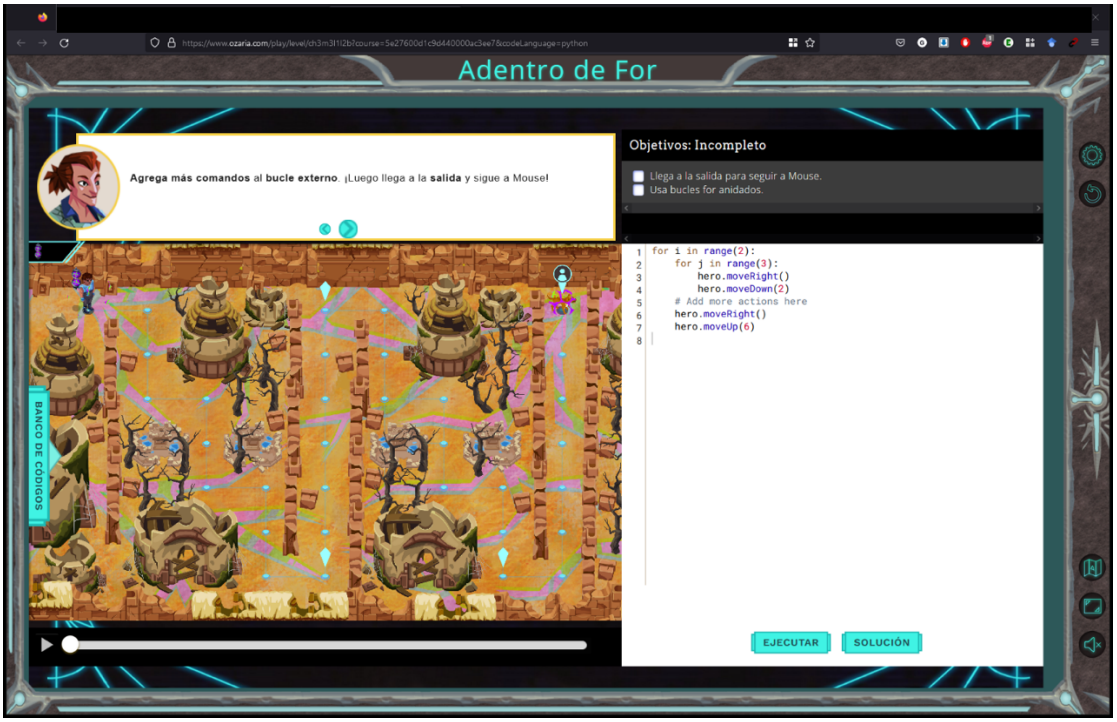
Ambos códigos, escribiendo el primero completo, producen el mismo resultado:

```
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
```

3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100

Y, estrictamente hablando, ambos códigos hacen el mismo número de operaciones, sin embargo, es claro que el segundo es más compacto y con un poco de experiencia hasta es más fácil de entender.

La plataforma Ozaria provee varios ejemplos más visuales e intuitivos.



Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Notas para el profesor impartidor del tema:

4. Anidaciones híbridas

Algo muy lógico es preguntarse si es posible utilizar ciclos while dentro de ciclos for o viceversa, y la respuesta es sí, de hecho, este tipo de estrategia se conoce como **anidación híbrida.**

Utilizar condicionales, ciclos while o for dentro de un condicional o ciclo diferente es muy común y útil al momento de generar códigos que cumplan con nuestros requerimientos.

Notas para el profesor impartidor del tema:

5. Aplicaciones

Las aplicaciones de los procesos iterativos y todas sus variantes son muy diversas y se ilustran perfectamente en las actividades de la plataforma, por lo cual no se comentarán aquí.

Módulo 2

Introducción

La forma en la que el ser humano representa la información, la almacena y la utiliza para la toma de decisiones depende directamente, tanto de sus necesidades como de las herramientas con las que dispone para hacerlo. Inicialmente, el ser humano solo tenía la necesidad de cuantificar los alimentos que recolectaba, por lo cual, con los números enteros era más que suficiente.

Posteriormente, su cultura evolucionó y el hombre inventó el concepto de compartir o prestar, así la necesidad de representar una deuda dio origen a los números negativos, mientras que la capacidad de cortar o dividir los alimentos originó la necesidad de usar números fraccionarios.

Durante todo este tiempo, el humano utilizó sus dedos para cuantificar, generando una numeración **decimal**. Para facilitar la forma de representar las cantidades, el ser humano desarrolló una serie de símbolos o dígitos (0, 1, 2, ..., 9) que, dependiendo de la posición de cada símbolo dentro de una cantidad, representa su valor; la primera posición representa las unidades, la segunda las decenas, y así sucesivamente.

Este tipo de numeración o forma de representar cantidades se conoce como **numeración digital**, pues se utilizan dígitos en diferentes posiciones para

representar la información deseada.

De manera paralela, los filósofos, principalmente los griegos, comenzaron a tratar de entender los diferentes conceptos y elementos de la naturaleza humana, de entre varias corrientes, Aristóteles comenzó a descomponer la naturaleza humana en verdades básicas que podían ser solo ciertas o falsas, esta corriente filosófica sentó las bases para el procesamiento de información representada por cantidades que pueden tomar solo dos valores: verdadero o falso.

Hubo que esperar hasta la aparición del tubo de vacío y, posteriormente el transistor, para que las bases sentadas por el pensamiento filosófico fueran utilizadas en la representación y procesamiento de información dando origen a la era **digital.**

El término digital se asocia, hoy en día, de una manera muy simplista con la representación de la información a partir de dígitos binarios, esto debido a que los sistemas de cómputo actuales son, en su gran mayoría, **digitales binarios,** es decir, representan la información en forma de dígitos que pueden tomar solo el valor 0 o 1.

La primera computadora electrónica funcional (ENIAC) era una computadora digital decimal, que representaba la información con números del 0 al 9, fue el matemático húngaro-americano John von Neumann quien se dio cuenta de la ventaja de representar las cantidades en forma binaria y participó en el diseño y fabricación de la primer computadora electrónica-digital binaria, la MARK 1.

La razón por la que Neumann decidió que era más eficiente representar las cantidades en forma binaria fue y sigue siendo por la forma en la que las computadoras digitales binarias están construidas. Las unidades básicas de procesamiento de una computadora digital binaria eran los tubos de vacío y hoy en día los transistores, los cuales pueden visualizarse como interruptores que permiten o no el flujo de electrones a través de ellos.

Por lo tanto, un transistor puede tener un valor 0 si no permite el paso de los electrones y 1 si lo permite. La gran combinación de 0 y 1 en diferentes posiciones permite representar y procesar información de manera masiva y muy rápida por las computadoras actuales, lo que es conocido como la era digital, o más propiamente, la era **digital binaria**.

Notas para el profesor impartidor del tema:

6. Condicionales compuestos

Al entender por qué la información dentro de una computadora se representa a partir de 0 y 1 resulta muy natural utilizar todas las herramientas de razonamiento desarrolladas por los filósofos antiguos en cuestión de premisas binarias.

Los filósofos antiguos desarrollaron toda una metodología de razonamiento lógico estructurado a partir de simples premisas y combinaciones lógicas que, posteriormente, se aplicaron a las ciencias de la computación.

Dentro de las ciencias de la computación, los condicionales IF representan la operación básica para la toma de decisiones, las premisas o silogismos, y la combinación de estos condicionales a partir de operadores lógicos permite generar funciones mucho más complejas, de hecho, el microprocesador de una computadora es capaz de generar la función más compleja imaginable, a partir de combinaciones de únicamente tres operaciones lógicas básicas.

Las **operaciones lógicas básicas** son la unión, la intersección y la negación, cada una produce un resultado binario a partir de la combinación de condicionales básicos.

La intersección

La operación lógica **intersección** es más comúnmente conocida como el operador “Y” o “AND” en inglés y cumple básicamente la función indicada por su uso gramatical. Por ejemplo, una madre puede condicionar la asistencia de su hijo a una fiesta **si** termina la tarea; otra madre puede condicionar la asistencia **si** limpia su cuarto.

Si haces tu tarea = vas a la fiesta.
Si limpias tu cuarto = asistes a la fiesta.

Una mamá más estricta puede utilizar la operación intersección para generar un condicional compuesto:

Si haces tu tarea Y limpias tu cuarto = vas a la fiesta.

En este caso, el hijo debe cumplir satisfactoriamente con ambas acciones para asistir. Fallar en alguna de ellas o en ambas provocará que no pueda asistir a la fiesta.

Este razonamiento puede expresarse de una manera más gráfica y fácil de entender a través de una **tabla de verdad**, para la cual tomaremos las siguientes consideraciones:

Llamemos a la premisa “Si haces tu tarea” como la entrada “a” y “limpias tu cuarto” la entrada “b”, así la consecuencia “ir a la fiesta” es generada a partir de la intersección de ambas entradas “a&b”. Un valor de 0 representa falso, es decir, no cumplir con alguna de las condiciones o no tener autorización para asistir a la fiesta, mientras que 1 es verdadero e implica haber cumplido con la condición o poder asistir a la fiesta.

a	b	a&b
0	0	0
0	1	0
1	0	0
1	1	1

La unión

La operación lógica **unión** se conoce como el operador “O” u “OR” en inglés y cumple, básicamente, la función indicada por su uso gramatical. De nuevo, con el ejemplo para asistir a la fiesta, una madre más condescendiente puede establecer la siguiente condición compuesta:

Si haces tu tarea o si limpias tu cuarto = asistes a la fiesta.

Así, el joven podrá asistir a la fiesta en caso de cumplir con una u otra condición, e incluso ambas. La única situación en la que no podrá asistir es al no cumplir con ninguna.

La tabla de verdad para la unión es la siguiente:

a	b	a o b
0	0	0
0	1	1
1	0	1
1	1	1

Negación

La negación es la única operación lógica que afecta a solo una condición y, básicamente, invierte el valor de esta condición. Por ejemplo, la condición “si limpias tu cuarto” cambia por la condición si no limpias tu cuarto, así, el estado que valía 0 ahora vale 1 y viceversa.

La negación de una premisa se representa utilizando una línea sobre la premisa $\overline{limpiar} = no\ limpiar$ o con una tilde $limpiar' = no\ limpiar$.

a	a'
0	1
1	0

Parte de la confusión puede deberse a no identificar correctamente las partes de una oración, por ejemplo, la oración “si no limpias tu cuarto, no asistes a la fiesta” cuenta con una doble negación, sin embargo, estas negaciones están en partes diferentes de la oración.

La primera está en la condicional y la segunda en la consecuencia, por lo tanto, no hay ambigüedad en su interpretación.

Notas para el profesor impartidor del tema:

7. Lógica binaria

Una vez conocidas las operaciones lógicas básicas, es posible generar combinaciones entre estas para controlar el comportamiento de un sistema u obtener una salida específica deseada, por ejemplo, considera la guillotina eléctrica ilustrada a continuación.

Por seguridad, la computadora digital interna de la máquina no debe permitir el accionamiento de la cuchilla cuando los dedos del operador se encuentren en la trayectoria de la cuchilla, por lo tanto, el diseñador agrega un par de botones en cada extremo con la leyenda “cut” en ellos, así la cuchilla se accionará si el botón 1 **&** el botón 2 están activados, obligando al usuario a emplear ambas manos para activar los botones y así, asegurar que no se encuentran en la trayectoria de la cuchilla.

Podemos hacer más compleja esta condición de accionamiento para hacer más seguro el sistema, agregando un interruptor que indique si la cubierta transparente está levantada o no, así nuestras premisas serán a = botón 1 presionado, b = botón 2 presionado y c = cubierta levantada: de esta manera, la cuchilla se activará si el botón 1 está presionado & el botón 2 lo está & la cubierta no está levantada.

Toda esta descripción se puede expresar de forma más compacta a través de una función lógica.

Función lógica y tabla de verdad

Una función lógica es el conjunto de condiciones lógicas que generan una consecuencia verdadera, por ejemplo, la función lógica que expresa las combinaciones necesarias para activar la cuchilla del ejemplo anterior es a&b&c', o de una forma más compacta, abc’.

Otra forma de representar el comportamiento de un sistema lógico es a través de su tabla de verdad, que es un conjunto de filas y columnas donde se expresan todos los valores posibles que pueden tomar las condiciones y las consecuencias correspondientes para estas combinaciones en los condicionales.

La tabla de verdad para el ejemplo de la guillotina es la siguiente:

a	b	c	a&b&c'
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

El procedimiento para generar una tabla de verdad es el siguiente:

1. Generar la tabla con las posibles combinaciones para las *n* condiciones involucradas en nuestra función lógica.

Para el ejemplo de la guillotina n = 3 condiciones, el número de combinaciones posibles se calcula con la fórmula:

combinaciones = 2^n, para n = 3 tenemos $8 = 2^3$combinaciones

La forma más sencilla para generar todas las posibles combinaciones es generar 2^{n-1} ceros seguidos de 2^{n-1} unos en la primera columna. La segunda columna se alterna 2^{n-2} ceros con 2^{n-2} unos un par de veces. Este proceso se repite para 2^{n-3}, 2^{n-4} , etc. Hasta alcanzar el límite 2^0, en donde se alternará entre solo 1 cero y 1 uno.

a	b	c
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

En la siguiente columna agregamos la primera operación a&b que, por simplicidad, puede expresarse solo como ab.

a	b	c	ab
0	0	0	0

0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

A continuación, generamos la segunda operación c':

a	b	c	a&b	c'
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	0	0
1	0	0	0	1
1	0	1	0	0
1	1	0	1	1
1	1	1	1	0

Finalmente, evaluamos la última operación: a&b&c'.

a	b	c	a&b	c'	abc'
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	0	0	0
1	0	0	0	1	0
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	1	0	0

Notas para el profesor impartidor del tema:

8. Álgebra de Boole

El álgebra de Boole es una serie de teoremas que permite cambiar la forma de expresar una función lógica por una función equivalente. Esta función equivalente puede simplemente ser diferente, o bien, puede generar algún tipo de ventaja,

como reducir el número de operaciones lógicas, reducir el número de variables o, simplemente, aislar un término para evaluar su importancia en la función.

Función equivalente

Se dice que dos funciones son equivalentes cuando ambas comparten los mismos maxitérminos.

Maxitérminos

Los maxitérminos de una función son las combinaciones de condiciones lógicas individuales que generan una salida verdadera, por ejemplo, los maxitérminos de la función “a o b” son ab’ + a’b + ab, con a’b + ab = a’b o ab.

a	b	a+b	
0	0	0	a’b’
0	1	1	a’b
1	0	1	ab’
1	1	1	Ab

Como simple ejercicio ilustrativo, vamos a generar la tabla de verdad para la función ab’ + a’b + ab.

a	b	ab’	a’b	ab	ab’ + a’b + ab
0	0	0	0	0	0
0	1	0	1	0	1
1	0	1	0	0	1
1	1	0	0	1	1

Como puedes ver, cada término de la función genera una única salida verdadera y no hay términos repetidos, por lo tanto, el número de maxitérminos en los que se puede descomponer una función es igual al número de veces que la función original toma el valor de 1.

Ahora podemos decir que las funciones a+b y ab’ + a’b + ab son equivalentes, o bien, a+b = ab’ + a’b + ab, por lo tanto, debe haber una serie de teoremas y procedimientos que nos permita pasar de una expresión a otra. Estos teoremas y propiedades constituyen las bases del álgebra de Boole y se enuncian a continuación:

Primero, el álgebra de Boole debe cumplir con las propiedades de:

1. Conmutatividad.

El orden de los operandos no altera la operación:

$$x+y=y+x \qquad xy=yx$$

2. Elemento neutro.

Existe un elemento que no altera la operación:

$$x+0=x \qquad x*1=x$$

3. Distributividad.

$$x(y+z)=xy+xz \qquad x+yz=(x+y)(x+z)$$

4. Asociatividad.

$$x+(y+z)=(x+y)+z \qquad x(yz)=(xy)z$$

5. Elemento complementario.

La operación con el complemento resulta en el término neutro:

$$x+x'=1 \qquad xx'=0$$

Algunas de estas propiedades dan como consecuencia los siguientes **teoremas:**

Teorema 1. Idempotencia

La operación de este término con el mismo resulta en el término original.

$$x+x=x \qquad x*x=x$$

Demostración:

x	x	x+x
0	0	0
1	1	1

x	x	x*x
0	0	0
1	1	1

Observa que, aunque se tienen dos operandos, ambos son el mismo, por lo tanto, solo podemos tener $2^1 = 2$ combinaciones.

Teorema 2. Identidad

La operación por el elemento nulo (o identidad) tiene como resultado el mismo elemento nulo.

Conviene recordar que el elemento nulo o identidad depende de cada operador, así, para la operación unión, el elemento identidad es el 1 y para la operación intersección es el 0.

x+1=1 x*0=0

Demostración:

	1	x+x
0	1	1
1	1	1

x	0	x*x
0	0	0
1	0	0

Teorema 3. Absorción

Cuando una variable aparece en ambos operandos, el resultado es la misma variable.

x+xy=x x*(x+y)=x

Demostración:

Se puede demostrar usando la tabla de verdad:

x	Y	xy	x+xy
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

x	Y	x+y	x(x+y)
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1

También podemos comenzar a usar los teoremas anteriores y simplificar las funciones mediante álgebra.

x+xy = x(1+y) (por idempotencia)

x(1+y) = x(1) (por identidad)

x(1) = x (por identidad)

De forma similar:

x(x+y) = x(1(1+y))

x(1(1+y)) = x(1(1))

x(1(1)) = x

Es muy importante mencionar que pese a la similitud que existe con el álgebra elemental (el que comúnmente conocemos), el álgebra de Boole tiene sus propias

reglas y teoremas que no siempre son análogos a los del algebra elemental, por lo tanto, las propiedades a las que estamos acostumbrados no siempre son válidas y hay que tener cuidado.

Teorema 4. DeMorgan

La negación de una operación es igual a aplicar el operador opuesto con los operandos negados.

(x+y)'=x'y' (xy)'=x'+y'

Demostración:

x	Y	(x+y)'	x'y'
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

x	Y	(xy)'	x'+y'
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Finalmente, el uso de estas propiedades y teoremas permite expresar funciones equivalentes, como ejemplo, volveremos al caso de los maxitérminos de la función OR.

ab+a'b+ab'

Aplicando distributividad podemos reescribir la función como:

b(a+a')+ab' = b(1)+ab' = b + ab'

Aplicando distributividad, de nuevo, podemos reescribir la función como:

b + ab' = (b+a)(b+b') = (b+a)(1) = b + a

Todas las expresiones anteriores son equivalentes y, dependiendo de la finalidad, pueden ser más o menos convenientes. He aquí la importancia de desarrollar la habilidad e intuición necesaria para identificar términos equivalentes y las propiedades o teoremas que se pueden aplicar para su conversión.

Notas para el profesor impartidor del tema:

9. Mapas de Karnaugh

Los mapas de Karnaugh son representaciones gráficas donde se localizan los maxitérminos de una función lógica. El hecho de ser una distribución gráfica permite, en muchas ocasiones, identificar de forma visual y mucho más sencilla los términos que comparten elementos en común y que, por lo tanto, es probable poder simplificar estos términos.

La metodología de mapas de Karnaugh incluye una serie de pasos y reglas que sistematizan la creación de los mapas, la identificación de términos simplificables y la forma de simplificarlos. Esta metodología, aunque válida para cualquier número de variables, se aplica principalmente para funciones con 2, 3 y hasta 4 variables; para el caso de 5 o más variables, el tamaño de los mapas y las posibles interacciones entre términos hace poco eficiente y dificulta mucho su aplicación.

El primer paso consiste en la creación del mapa, es decir, generar la distribución gráfica de los maxitérminos. Dependiendo del número de variables, esta distribución es una matriz de 2x2, 2x3 o 4x4, donde cada posición de esta matriz representa una posible combinación de términos:

	a	a'
B		
b'		

	ab	a'b	a'b'	ab'
C				
c'				

	ab	a'b	a'b'	ab'
Cd				
c'd				
c'd'				
cd'				

Cada celda de este arreglo representa la combinación de los términos de la fila y la columna correspondiente:

	a	a'
B	ab	a'b
b'	ab'	a'b'

	ab	a'b	a'b'	ab'
C	abc	a'bc	a'b'c	ab'c
c'	abc'	a'bc'	a'b'c'	ab'c'

	ab	a'b	a'b'	ab'
Cd	abcd	a'bcd	a'b'cd	ab'cd

c'd	abc'd	a'bc'd	a'b'c'd	ab'c'd
c'd'	abc'd'	a'bc'd'	a'b'c'd'	ab'c'd'
cd'	abcd'	a'bcd'	a'b'cd'	ab'cd'

A continuación, se deben localizar los maxitérminos de la función dentro de este arreglo, por ejemplo, el mapa de la función a'bc'd + a'b'c'd' + a'b'c'd + a'b'c'd' es el siguiente:

	1	1	
	1	1	

Finalmente, se localizan conjuntos de 2, 4 u 8, unos dentro del mapa y el resultado final son los términos comunes de estos arreglos. Para el ejemplo indicado se tiene una sola agrupación de 4 unos y los únicos términos comunes entre estos elementos son: a'c', así, podemos decir que a'bc'd + a'b' c'd' + a'b'c'd + a'b'c'd' = a'c'.

	1	1	
	1	1	

a'bc'd + a'b'c'd' + a'b'c'd + a'b'c'd'

Es claro que, para el ejemplo anterior, los elementos comunes pueden identificarse de forma directa de la función original, sin embargo, para funciones más complejas, la distribución gráfica ayuda mucho a la identificación de grupos con términos comunes.

Notas para el profesor impartidor del tema:

10. Aplicaciones

Una vez más, las aplicaciones sobre los condicionales compuestos y sus simplificaciones se encuentran ejemplificadas en los ejercicios de la plataforma Ozaria.

Notas para el profesor impartidor del tema:

11. Funciones

El término **función** se utiliza en varias áreas del conocimiento. Una función matemática establece una relación entre dos conjuntos, mientras que una función lógica establece las condiciones lógicas necesarias para tener un resultado verdadero.

De igual forma, dentro de las ciencias de la computación, el término “función” tiene una definición muy específica. Básicamente, una función es un conjunto de instrucciones que realizan un procedimiento específico para una serie de parámetros o valores perfectamente identificados. Si bien, dentro de un programa esto es algo muy común, la particularidad de la función radica en qué tan continuamente se necesita repetir todo este procedimiento.

Por ejemplo, supongamos que se crea un programa que almacena el nombre, dirección y edad de un usuario y, finalmente, responda con un saludo personalizado. El procedimiento es bastante sencillo:

Pseudocódigo:

Inicia un contador n=1

Crea una variable del tipo lista llamada nombres.

Crea una variable del tipo lista llamada direcciones.

Crea una variable del tipo lista llamada edades.

Pregunta al usuario por su nombre y almacena la entrada en la variable nombres en el índice n.

Pregunta al usuario por su dirección y almacena la entrada en la variable direcciones en el índice n.

Pregunta al usuario por su edad y almacena la entrada en la variable edades en el índice n.

Escribe “Hola:” nombres[n]

Como puedes ver, este código es muy simple, sin embargo, el hecho de no conocer el número preciso de individuos que se van a registrar hace complicado el colocar este código dentro de un bucle para repetirlo de manera indefinida.

Nota que sí es posible colocarlo en un ciclo while, donde la condición de repetición sea preguntarle al usuario si hay más registros pendientes, sin embargo, consideremos que este programa es parte de uno mucho más grande, donde podamos ver el número de registros hasta el momento, verificar los datos de un registro específico, buscar registro por nombre, etc.

Esto dificulta mucho más la implementación de un bucle que repita el proceso de registro cada vez que sea necesario. La solución más lógica e intuitiva es apartar el código específico para el registro y utilizarlo cada vez que sea necesario, esto justamente es una función, se le provee un nombre al conjunto de este código, se almacena de manera separada y se manda a llamar cada vez que se necesite.

De esta manera se reducen significativamente las líneas de código necesarias para repetir el proceso una y otra vez.

La manera de generar una función en Python es la siguiente:

Se debe comenzar con el comando “def”, que le indica a Python que se va a definir una función. Enseguida se coloca el nombre que deseamos para esta función; hay que tener cuidado de no duplicar nombres o utilizar nombres ya definidos dentro del mismo lenguaje.

Dentro de un paréntesis se colocan todos los parámetros que el usuario debe proveer para llevar a cabo el procedimiento; el número de parámetros depende directamente de la finalidad de la función.

Puede haber funciones donde no se requiera ningún parámetro o funciones donde se necesiten varios de ellos. Finalmente, se colocan dos puntos y en las siguientes líneas se agregan las instrucciones que forman a la función.

No olvides respetar el indentado de pertenencia. Al final, si la función debe regresar uno o varios resultados se utiliza la instrucción “return”, seguido de la variable o variables cuyos valores debe regresar la función.

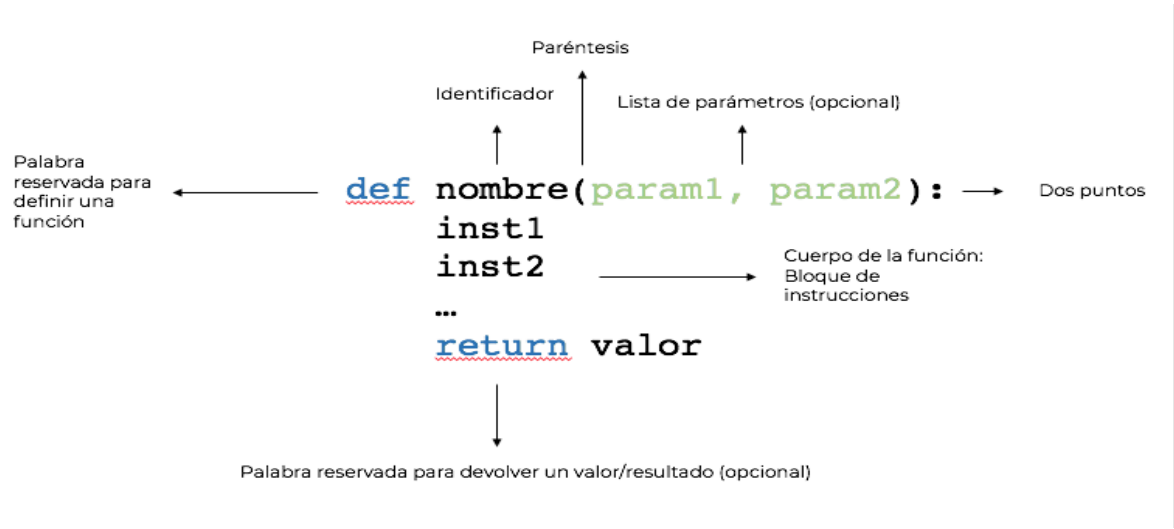


Figura 1. Generar una función en Python.

Por ejemplo, la función jumpArround creada en la actividad *Pi jump* no tiene ningún parámetro de entrada ni de salida y solo hace que "*noodles*" salga hacia arriba, luego a la izquierda y luego hacia abajo:

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Módulo 3

Introducción

La razón principal para aprender a programar y codificar en cualquier lenguaje es generar programas y aplicaciones que ayuden a las personas a simplificar tareas o resolver problemas, más allá de crear nuevas dificultades para ellos.

Por lo tanto, la forma en la que el programador planea que el usuario va a interactuar con sus programas es de vital importancia y vale mucho la pena comenzar a formar en los jóvenes un criterio y estándares de funcionalidad, operatividad y simplicidad que le permitan planificar y prever las necesidades, deficiencias y situaciones especiales con las que el usuario se podrá encontrar durante el uso de sus programas, para así poder ayudarlo, entenderlo y facilitar su experiencia.

Notas para el profesor impartidor del tema:

12. Interacción con el usuario

Estrictamente hablando, una interfaz de usuario es cualquier medio por el cual el usuario puede interactuar con algún elemento tecnológico, por lo tanto, esto abarca elementos físicos y no físicos. Por ejemplo, un teclado, un micrófono o una pantalla táctil son interfaces físicas de **hardware**, mientras que los menús, las pantallas, las barras de herramientas, etc. son interfaces de **software** y, por supuesto, siempre pueden existir elementos que comparten ambas características de **software-hardware**, por ejemplo, los botones de una aplicación que interactúan con la pantalla táctil, pero son diseñados desde el código.

Ambos tipos de interfaces son muy importantes para el producto final. Un diseño ergonómico y funcional en ambos tipos de interfaces puede ayudar mucho a que un programa en verdad resuelva un problema o simplemente se convierta en un problema más grande.

Los expertos en diseño de productos, diseño industrial, marketing, etc. se especializan en las características físicas de una interfaz y, aunque también existen expertos encargados de vigilar la ergonomía y la funcionalidad de las interfaces de software, siempre es importante que un programador tenga buenos hábitos de diseño que faciliten esta labor.

Es importante resaltar que, para nuestro caso, el área de interés son las interfaces de software y, con esto en mente, consideramos necesario que el alumno desarrolle una empatía con el usuario, que entienda la forma en que este va a interactuar con sus programas y que siempre piense en las necesidades de los usuarios al momento de diseñar su interfaz, buscando que sea ergonómica y funcional.

Notas para el profesor impartidor del tema:

13. Interfaz de usuario

La interfaz de usuario o UI *(user interface)* es el conjunto de instrucciones, funciones y procedimientos generados de manera específica para interactuar con el usuario. Si bien, un programa puede estar perfectamente generado para funcionar y cumplir su propósito, es posible que se requiera de conocimiento muy especializado, ya sea del tema en específico del programa, o bien, que requiera de un dominio en conceptos propios de la programación.

Es responsabilidad del programador el entender el nivel de conocimiento y dominio de conceptos, tanto de los temas del programa como de programación, así como del sector del público al que su programa va dirigido y hacer todo lo posible para subsanar o reducir las posibles deficiencias del propio usuario.

Por ejemplo, hoy en día no es necesario que una persona sepa de realidad aumentada ni visión artificial, mucho menos de la memoria o procesador de su dispositivo para que esta pueda utilizar los filtros con la cámara de su celular y divertirse sin preocuparse por la iluminación, el tiempo de exposición, la capacidad de la GPU, etc. Esto es gracias a que los programadores entendieron todo esto y simplificaron el proceso para el usuario.

De manera específica, la interfaz de usuario son todas las instrucciones que reciben o proveen información del o al usuario. Ahora es muy común que esta interacción se haga mediante elementos gráficos, pues los seres humanos tenemos mucha mayor afinidad a los elementos visuales y llamativos, así varios programas tienen una **interfaz gráfica de usuario o GUI**, por sus siglas en inglés.

En el reto final de este curso, el alumno desarrollará su propia interfaz gráfica usando las herramientas generadas por el proveedor. A continuación, se presenta otro ejemplo de una buena interfaz, lo cual facilita mucho el proceso y le permite al alumno enfocarse únicamente en la parte principal:

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

Claridad

Una buena interfaz transmite la información de manera precisa para evitar que el usuario cometa errores de interpretación durante la interacción.

Concisión

Una buena interfaz provee solo la información que el usuario necesita y no lo distrae con información inútil o poco relevante.

Coherencia

Esta característica es la que hace intuitiva una interfaz, es decir, si una acción generó una respuesta bajo una situación, se espera que la misma acción genere una reacción similar en otras circunstancias, permitiéndole a la persona crear patrones de uso de manera sencilla y práctica.

Flexibilidad

Es permitirle al usuario modificar o personalizar la forma de interactuar. Ayuda mucho a reducir el tiempo de aprendizaje que el usuario requiere para aprender a usar nuestro programa.

Atractivo visual

Siempre es importante tener un equilibrio entre los detalles visuales y la concisión. Los elementos visuales muy llamativos pueden atraer la atención del usuario a detalles poco relevantes y distraerlo de los elementos que sí requieren de su atención.

Si bien, una GUI es muy útil para simplificar la interacción con el usuario, no es la única forma de interactuar con este, además de que generar gráficos interactivos en un lenguaje de programación suele ser una de las habilidades más difíciles de adquirir, ya que requiere de mucho tiempo y experiencia.

Notas para el profesor impartidor del tema:

14. Ergonomía y funcionalidad

En agosto de 2000, el Consejo de la Asociación Internacional de Ergonomía (IEA) acordó una definición que ha sido adoptada como “oficial” por muchas entidades, instituciones y organismos de normalización.

Para el caso específico mexicano en la norma oficial mexicana NOM-036-1-STPS-2018:

“Ergonomía (o estudio de los factores humanos) es la disciplina científica que trata de las **interacciones entre los seres humanos y otros elementos de un sistema**, así como la profesión que aplica teoría, principios, datos y métodos al diseño con objeto

de optimizar el bienestar del ser humano y el resultado global del sistema" (Diario Oficial de la Federación, 2018).

La ergonomía tiene en consideración factores físicos, cognitivos, sociales, organizacionales y ambientales, pero con un enfoque "holístico", en el que cada uno de estos factores no deben ser analizados aisladamente, sino en su interacción con los demás.

Para el caso específico de los sistemas computacionales, la ergonomía se enfoca en las características y capacidades físicas de los seres humanos que interactúan con sistemas de cómputo. Dichas características humanas pueden dividirse en tres grupos principales:

1. **Limitaciones físicas:** no todos los usuarios son iguales físicamente. Por ejemplo, si la interfaz está diseñada solo para individuos diestros o con cierta altura, muchos tendrán problemas para usarla y se verán afectados.
2. **Habilidades cognitivas:** no todos los usuarios son expertos o tienen una alta habilidad cognitiva, por lo que no debería ser necesario manejar mucha información para usar una interfaz. Por ejemplo, muchos sistemas utilizan el término dirección en lugar de URL para facilitar su comprensión.
3. **Necesidades emocionales:** la interfaz no debe confundir al usuario, ni sobre cómo empezar a usarla ni cuando ocurre un error. La confusión lleva a la frustración, la que eventualmente se transforma en estrés, produciendo un daño emocional en el largo plazo.

Debido a estas características, los mayores desafíos que enfrenta la ergonomía están relacionados con la diversidad de los usuarios. Algunos aspectos son bastante controlables y medibles, como la edad y los hábitos, pero otros son muy difíciles de manejar. Las expectativas y el nivel de conocimiento de las personas son variados y su investigación es compleja.

Esta variedad en los posibles usuarios también dificulta evaluar la ergonomía de un sistema o interfaz, por lo cual se han desarrollado herramientas especializadas, como los *tests* de usuario y las herramientas de usabilidad, con la finalidad de evaluar y cuantificar las ventajas de un sistema.

Es claro que cuando se habla de cuestiones estéticas o visuales existen subjetividades que pueden dificultar el diseño de una buena UI, sobre todo cuando se trabaja en equipos interdisciplinarios o interculturales.

Por lo tanto, algunos expertos se han dado a la tarea de definir parámetros internacionales y mundiales sobre las buenas prácticas para desarrollar interfaces de usuario, quizás, el estándar más conocido y seguido es la norma ISO 9241, que se

centra en la ergonomía de la interacción entre la persona y el sistema, específicamente, en aspectos como la facilidad de la comunicación y el dinamismo.

Cabe mencionar que estos estándares no son subjetivos o dictados de forma arbitraria, son el resultado de investigaciones, estudios y pruebas de campo de los diferentes elementos que pueden componer una UI.

Dentro de una página web, aplicación, programa o cualquier herramienta digital, el diseño de la interfaz del usuario debe garantizar que esta sea capaz de adecuarse a la tarea, contar con tolerancia a los errores, ser capaz de personalizarse, proveer suficiente nivel de control, adecuarse al aprendizaje, ser autodescriptiva y, sobre todo, dar conformidad con las expectativas del usuario.

El diseño de una interfaz de usuario se ayuda de muchas otras áreas del conocimiento, por ejemplo, del diseño gráfico, de la psicología, etc.

Por lo que una buena UI respeta los principios de coherencia y calidad gráfica, como los códigos de colores para apoyar las tareas de los usuarios, minimalismo, distribución, Gestalt y estandarización.

Notas para el profesor impartidor del tema:

15. Producto final

Finalmente, la conjunción de algoritmos e interfaces de usuario generan el producto final. Si bien, el éxito de este depende de muchas variables fuera del control de los diseñadores y programadores, hay elementos básicos que se deben buscar respetar para aumentar la probabilidad del éxito.

Consistencia

El diseño debe ser consistente a través de toda la aplicación, tanto en las secuencias de acciones como en las terminologías y las convenciones de la plataforma. Esto ayuda a las personas a reconocer elementos y entender su jerarquía y utilidad.

Eficiencia

La interfaz debe permitir el uso eficiente del tiempo del usuario, cargando y presentando el contenido dentro un tiempo aceptable. Mientras más haces esperar a una persona, más estrés se acumula en ella.

Existen ciertos rangos para las acciones, cada uno le comunica al usuario que está ocurriendo algo distinto. También debes incluir funcionalidades para usuarios avanzados, por ejemplo, aceleradores o atajos.

Diseño

Idealmente, el diseño debe ser atrayente. Esto ayuda al usuario a consumir los datos y minimizar el estrés, logrando el efecto de "sentirse bien". Los principios de contraste, repetición, alineación y proximidad son naturales para los humanos y son percibidos como familiares.

Memoria

El uso de la memoria de los usuarios debe minimizarse. Toda la información que se necesita para realizar una tarea debe presentarse o requerirse de forma simple.

Ayuda contextual

Los recursos de ayuda deben ser visibles y estar siempre disponibles, sin interrumpir la navegación.

Estructura y espacio

La interfaz debe considerar la posición de las manos, la movilidad de los dedos y el campo visual de los usuarios en computadoras de escritorio, tabletas y *smartphones*. Asimismo, los botones, campos y menús desplegables deben ser fáciles de activar en los distintos dispositivos.

Además, es importante ofrecer *feedback* visual sobre el estado de los elementos, indicando si están activos, presionados, seleccionados o cargando, entre otros. Esto le informa al usuario que el sistema recibió su orden y está generando una respuesta.

Una de las claves para poner la tecnología al servicio de las personas es que las interfaces estén diseñadas pensando en ellas. Todas las funcionalidades y ventajas que ofrece el Internet y las plataformas digitales no pueden ser aprovechadas si los usuarios no son capaces de usar los sitios de forma óptima.

Para concluir, revisa en la plataforma Canvas la metodología, el temario y la evaluación a la que corresponde la entrega de cada reto, así como las políticas del curso para la modalidad semestral o tetramestral; hay una pestaña para cada una dentro de Evaluación.

También, desde la plataforma Canvas se encuentra la sección Retos, donde se revisarán las instrucciones de cada uno de ellos, así como sus entregables.

Las calificaciones se subirán en la plataforma Canvas.

Notas para el profesor impartidor del torneo (aplica solo en versión semestral):

A continuación, se presentan las indicaciones e instrucciones para la realización del torneo:

Objetivo

- Fomentar el aprendizaje y la aplicación de tecnologías avanzadas en la resolución de problemas matemáticos, promoviendo la colaboración, la creatividad y el espíritu competitivo entre los estudiantes.

Selección de participantes

- El primer lugar de cada grupo (o los dos primeros lugares en el caso de campus pequeños) del Reto Final de la clase de Tecnologías de la Información II e Information Technologies II será seleccionado para participar en el torneo.

Previo al torneo

- Los estudiantes tienen la oportunidad de realizar mejoras a sus aplicaciones antes del torneo.
- Los docentes deben evaluar el reto final para seleccionar a los estudiantes que representarán al grupo.
- El campus, mediante el Líder de Docentes, definirá a los profesores que conformarán el panel de jueces.

Desarrollo de torneo

- Los estudiantes participantes presentarán sus aplicaciones ante un panel de jueces compuesto por docentes de Tecnologías y Matemáticas, seleccionados por el campus.
- Cada presentación incluirá una demostración en vivo de la aplicación, una explicación del proceso de desarrollo y una sesión de preguntas y respuestas.

Criterios de evaluación

- Se evaluará utilizando la misma rúbrica del Reto final: Funcionamiento, Robustez, Versatilidad, Interfaz de usuario y Aspecto estético.

Premiación

- El estudiante que obtenga la mayor puntuación será el ganador del torneo en el campus y recibirá una insignia digital, además de reconocimiento en las redes sociales del campus (previa autorización para el uso de su imagen).

Para conocer la metodología, fechas de cuándo se implementa el reto final y el torneo (solo en semestral), así como las instrucciones, haz clic **aquí**.

Bibliografía

Diario Oficial de la Federación. (2018). *NORMA Oficial Mexicana NOM-036-1-STPS*. Recuperado de https://dof.gob.mx/nota_detalle.php?codigo=5544579&fecha=23/11/2018#gsc.tab=0