



# Introducción al Pensamiento Computacional - Semana 10



© Todos los derechos reservados Universidad Rafael Landívar URL.

---

## DESEMPEÑOS ESPERADOS

- ≡ Desempeños esperados

---

## DESARROLLO DE CONOCIMIENTOS

- ≡ Encontrar patrones en programas
- ≡ Abstracciones en programación

---

## APLICANDO LO APRENDIDO

- ≡ Actividad 1
- ≡ Actividad 2
- ≡ Actividad 3

---

## Actividad 4

### DE LA TEORÍA A LA PRÁCTICA Y REFLEXIÓN

---

#### Recursos

### CRITERIOS DE EVALUACIÓN

---

#### Rúbrica de evaluación

#### Diario de experiencias de laboratorio

### FUENTES DE REFERENCIAS

---

#### Referencias

### CRÉDITOS

---

#### Créditos

# Desempeños esperados

---



## El estudiante:

- 1 Aplica principios simples para encontrar patrones en los programas.
- 2 Analiza cómo se aplica la abstracción en el contexto de la programación.
- 3 Identifica tipos de datos propios para usar en las soluciones.
- 4 Integra clases como forma de generar sus propios tipos y abstracciones.

5

Examina arreglos de patrones ya existentes.

## Encontrar patrones en programas

---

# Encontrar patrones en programas

Encontrar patrones durante la descomposición de un problema se mencionó en el módulo 3 "patrones y generalización". Es muy importante buscar los patrones en un problema. Cuando se realiza el proceso de descomposición se irán identificando conceptos que comparten similitudes: procesos que trabajan de manera próxima y entidades que juntas forman una sola abstracción.

Esta búsqueda de patrones empieza durante el trabajo de diseño, pero no termina al implementar el software. Al escribir el software se pueden identificar patrones para aprovechar sus ventajas.

"Cada parte funcional de un programa debería ser implementada en una sola parte del código fuente. Si hay funciones similares, pero son algunas variantes de código, resulta beneficioso combinarlas abstrayendo las partes", Pierce (2002).

Este consejo aplica para dos principios básicos:

1

No se repita así mismo.

2

Encuentre lo que varía y encapsúlelo.

Aplicaremos estos principios en el ejemplo del dibujo de la carita feliz estudiada anteriormente, como se muestra a continuación.

```
def render_neutral_face():
    core_draw.draw_oval(50, 50, 120, 120) # Face
    core_draw.draw_oval(60, 60, 80, 80) # Left eye
    core_draw.draw_oval(65, 65, 75, 75)
    core_draw.draw_oval(90, 60, 110, 80) # Right eye
    core_draw.draw_oval(95, 65, 105, 75)
    core_draw.draw_line(70, 110, 100, 110) # Mouth
```

La función "render\_neutral\_face()" llama a la función "draw\_oval" muchas veces seguidas, sin embargo, las llamadas no son idénticas, ya que los parámetros varían.

Siguiendo los consejos anteriores, no se repita a sí mismo.

```
def render_neutral_face():
    for something in collection_of_somethings:
        core_draw.draw_circle(something)
    core_draw.draw_line(70, 110, 100, 110)
```

Luego, encuentre lo que varía y encapsúlelo. Los parámetros de círculo varían, así que se pueden encapsular en su propia colección.

```
def render_neutral_face():
    circle_dimensions = [
        (50, 50, 120, 120), (60, 60, 80, 80), (65, 65, 75, 75),
        (90, 60, 110, 80), (95, 65, 105, 75)
    ]
    for d in circle_dimensions:
        core_draw.draw_circle(d[0], d[1], d[2], d[3])

    core_draw.draw_line(70, 110, 100, 110)
```

Se puede invocar a la solución para dibujar diferentes caras.

```
def render_mouthless_face(circle_params):
    for param in circle_params:
        core_draw.draw_circle(param[0], param[1], param[2], param[3])

def render_neutral_face():
    circle_params = [
        (50, 50, 120, 120),
        (60, 60, 80, 80), (65, 65, 75, 75),
        (90, 60, 110, 80), (95, 65, 105, 75)
    ]
    render_mouthless_face(circle_dimensions)
    core_draw.draw_line(70, 110, 100, 110)

def render_smiley_face():
    circle_dimensions = [
        (50, 50, 120, 120),
        (60, 60, 80, 80), (65, 65, 75, 75),
        (90, 60, 110, 80), (95, 65, 105, 75)
    ]
    render_mouthless_face(circle_dimensions)
    core_draw.draw_arc(30, 70, 40, 10, 0, -180)
```



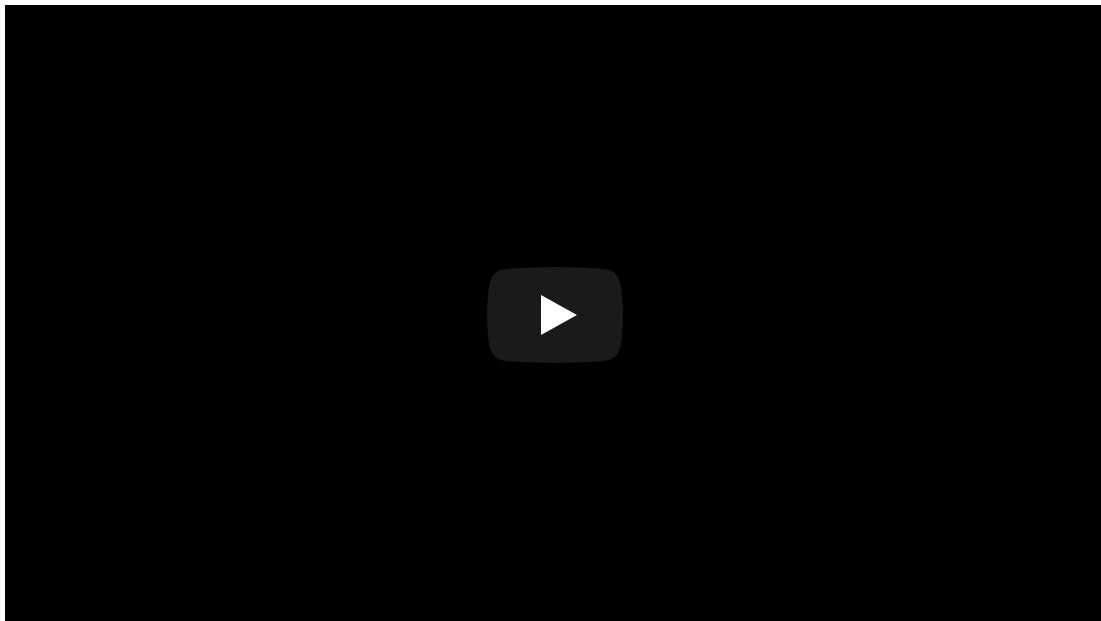
## Abstracciones en programación

---



Las abstracciones fueron estudiadas en el módulo 4. El encontrar patrones ayuda a las abstracciones. Abstraer significa expresar una idea en un cierto contexto y suprimir algunos detalles que son irrelevantes para dicho contexto.

Los lenguajes de programación son una abstracción en sí mismos. Las computadoras no comprenden el código que Ud. escribe. El código es transformado en lenguaje de máquina (ceros y unos) antes de que se ejecute. La programación abstrae la cantidad de ceros y unos y le permite enfocarse en escribir una solución. Si quisiera ampliar sobre este tema, puede consultar:



## How do computers read code?

When you first learned to write code, you probably realized that computers don't really have any common sense. You need to tell a computer exactly what you w...

[VER EN YOUTUBE >](#)

Sin embargo, en el contexto de curso, la abstracción se refiere a las clases y tipos que Ud. creará como parte de su solución.

## Tipos de datos nativos

Para que la solución funcione, se deberán almacenar piezas de información, según su naturaleza podrá ser almacenada en diferentes tipos de datos. Por ejemplo, para almacenar un número, se requiere un valor entero, pero para almacenar un texto se requiere una cadena de caracteres (string). Los lenguajes de programación ofrecen tipos de datos nativos. En el caso de C# se estudiaron en módulos anteriores, aquí nuevamente:



## Built-in types - C# reference

Learn C# built-in value and reference types

[MÁS INFORMACIÓN MICROSOFT >](#)

The following table lists the C# built-in value types:

C# type keyword	.NET type
bool	System.Boolean
byte	System.Byte
sbyte	System.SByte
char	System.Char
decimal	System.Decimal
double	System.Double
float	System.Single
int	System.Int32
uint	System.UInt32
nint	System.IntPtr
nuint	System.UIntPtr
long	System.Int64
ulong	System.UInt64
short	System.Int16
ushort	System.UInt16

The following table lists the C# built-in reference types:

C# type keyword	.NET type
object	System.Object
string	System.String
dynamic	System.Object

Es importante saber los límites de almacenamiento de estos valores para evitar errores en los programas.

## Creando los propios tipos de datos

Si recordamos el ejemplo de la compañía de renta de autos, se manejaban vehículos de tipo carro, camioneta y motocicleta. Todos estos eran versiones de un concepto más general llamado vehículo. Cada tipo soportaba características específicas como la cantidad de combustible y operaciones, como el registro y reporte de millas.

Si los tipos primitivos que ofrece un lenguaje no proveen lo que requiere una solución, se pueden realizar los propios tipos. Para ello se aplica la programación orientada a objetos.

Un tipo es como una plantilla. Un objeto es la instancia de un tipo, con base a una plantilla. Un objeto pertenece a una sola plantilla, pero una plantilla puede tener tantos objetos como quiera. Para crear su propio tipo, se debe escribir una plantilla que le indique a la computadora cómo se debe comportar.

En estos videos se explican los conceptos de clases y objetos:

 YOUTUBE



## Clases y Objetos - Programación Orientada a Objetos

Si te gusta mi contenido  invítame un café ☕

<https://www.buymeacoffee.com/fredygeek>/Encuentra este video y más en:<https://fredygeek.com/2020/09/06/clases-...>

**VER EN YOUTUBE >**

 YOUTUBE



## Object Oriented Programming 2 - Classes and Objects

Suggest new or help me make more videos here:

<http://patreon.com/opencanvasln> In this lesson we delve into what is a Class and Object is in Object Oriented Prog...

**VER EN YOUTUBE >**

 YOUTUBE



## Fundamental Concepts of Object Oriented Programming

This video reviews the fundamental concepts of Object Oriented Programming (OOP), namely: Abstraction, which means to simplify reality and focus only on the ...

[VER EN YOUTUBE >](#)

## Minecraft

 YOUTUBE





## La Lógica de la Programación Orientada a Objetos explicada con Minecraft

Les traemos el inicio de este tema, a partir de este vídeo ya podemos traer diferentes vídeos de en 2 minutos, así que preparateInstagram:

<https://www.instagram.com/>

**VER EN YOUTUBE >**

**En C# los tipos se llaman clases.**

 MICROSOFT



### Clases

Un tipo que se define como es un tipo . En tiempo de ejecución, cuando se declara una variable de un tipo de referencia, la variable contiene el valor hasta que se crea explícitamente una instancia de la clase mediante el operador o se le asigna un objeto de un tipo compatible que se puede haber creado en otro lugar, como se muestra en el ejemplo siguiente: //Declaring an object of type MyClass.

[MÁS INFORMACIÓN MICROSOFT >](#)

**Se recomienda tomar este curso de codeacademy sobre clases:**

[!\[\]\(51423b03ed5dbe39f78a50141211e114\_img.jpg\) CODECADEMY](#)



## **Learn C#: Classes and Objects | Codecademy**

Define your own custom types using classes.

[MÁS INFORMACIÓN CODECADEMY >](#)

**Las relaciones entre las clases pueden ser de herencia y polimorfismo. En estos videos se explica.**

 YOUTUBE



## **Herencia y Polimorfismo | Programación Orientada a Objetos [Video 3]**

En este video vemos dos de las grandes funcionalidades que ofrece la programación orientada a objetos.Url del repo:<https://github.com/codin-eric/herencia-y-p...>

**VER EN YOUTUBE >**

## **Minecraft**

 YOUTUBE





## **La Lógica de HERENCIA explicada con Minecraft**

Continuamos con esta serie de Programación Orientada a Objetos explicado de la mejor manera. Conviértete en miembro del canal:

<https://www.youtube.com/channel...>

**VER EN YOUTUBE >**

**Se recomienda tomar este curso de codeacademy sobre interfaces y herencia:**

**CODECADEMY**



## Learn C#: Interfaces and Inheritance | Codecademy

```
interface IAutomobile { string LicensePlate { get; } double Speed { get; } int Wheels { get; } } // The IAutomobile interface has three properties. Any class that implements this interface must have these three properties.
```

[MÁS INFORMACIÓN CODECADEMY >](#)

## Patrones

Ya hemos visto cómo encontrar patrones en las soluciones y cómo convertirnos en propias generalizaciones y tipos. Conforme se adquiera experiencia, algunos patrones aparecerán repetidamente; con algunas diferencias en cada caso, pero con una estructura similar.

Existen muchos patrones que han sido encontrados y clasificados a través del tiempo, muchos de ellos se describen en libros de diseño orientado a objetos. Al familiarizarse con estos patrones, se tienen algunas ventajas, como:

- Son soluciones probadas para problemas comunes.

- Reducen la cantidad de esfuerzo para la programación.
- Al nombrarlos, proveen un vocabulario común para comprender las ideas en la solución.
- Ocultan ciertos detalles, permitiendo que se discuta la solución a un nivel más alto, en lugar de tratar con líneas individuales de código.

Los patrones pueden ser simples (con unas pocas líneas de código) o más complejos, incluyendo varias clases. Los patrones complejos o patrones de diseño pueden ser clasificados si son estructurales, creacionales y de comportamiento. Cada patrón resuelve un problema diferente, pero juntos tienen como objetivo el establecer un conjunto de soluciones reutilizables para los problemas de diseño comunes, simplificar los programas, agilizar el desarrollo y tener una terminología común.

## Patrones de diseño

Descargue las siguientes infografías sobre patrones de diseño:



**Patrones creacionales.pdf**

353.3 KB



**Patrones estructurales.pdf**

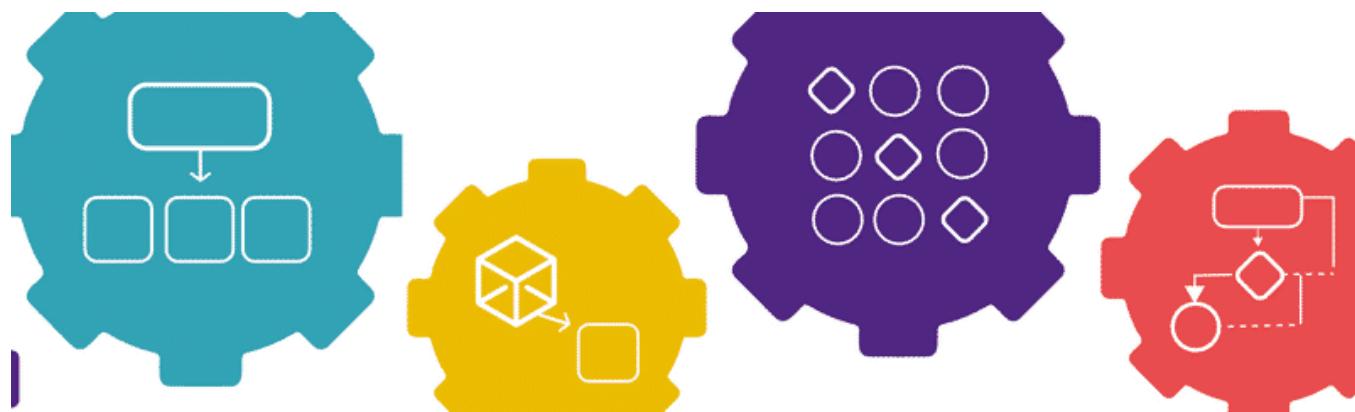
571.3 KB





**Patrones de comportamiento.pdf**

462.4 KB



## Actividad 1

---



The background of the slide features a close-up photograph of two interlocking gears. One gear is a solid blue color, and the other is a solid yellow color. They are shown from a perspective angle, highlighting their teeth and how they mesh together.

## Actividad 1



## Instrucciones

Marque si los siguientes enunciados son verdaderos o falsos.

Todas las variables en un lenguaje de programación deben ser del mismo tipo.

Verdadero



Falso

SUBMIT

Crear un objeto de una clase específica se llama instancia.

---



Verdadero



Falso

SUBMIT

Una subclase puede reescribir los métodos que hereda de su clase padre.

Verdadero

Falso

SUBMIT

Reutilizar patrones ayuda a reducir el esfuerzo al diseñar soluciones.

Verdadero

Falso

SUBMIT

Los patrones de diseño se clasifican en: creacionales, estructurales y de comportamiento.

---

Verdadero

Falso

SUBMIT

## Actividad 2

---



Actividad 2



## Instrucciones

Escriba un programa en C# que imprima las palabras únicas de una oración cualquiera. Asuma que la oración solo contiene letras y espacios (sin signos de puntuación).

---

Ayuda: ¿existen funciones para eliminar duplicados de listas en C#?

Diríjase a la plataforma para subir su actividad en el espacio correspondiente.

## Actividad 3

---



Actividad 3



Construir un juego de piedra, papel o tijeras. Primero, defina una clase para cada forma: piedra, papel, tijeras. Proporcione a cada clase un método que se llame “partida” que tome otra forma y devuelva si gana o pierde contra esta forma generada.

Diríjase a la plataforma para subir su actividad en el espacio correspondiente.

## Actividad 4

---



Actividad 4



## Instrucciones

Agregue una clase como un jugador de computadora. Se requiere un solo método que se llame “elegir”, que regrese una de las tres formas de forma aleatoria.



Ayuda: ¿existe una opción de función aleatoria/random en C#?

Diríjase a la plataforma para subir su actividad en el espacio correspondiente.



## Recursos

---



## Recursos

### Instrucciones:

Haber instalado el lenguaje de programación en las computadoras que utilizará para realizar las prácticas.



## Rúbrica de evaluación

---

Rúbrica





Descargue la siguiente rúbrica de evaluación.



**Rúbrica de Evaluación.pdf**

1635 KB



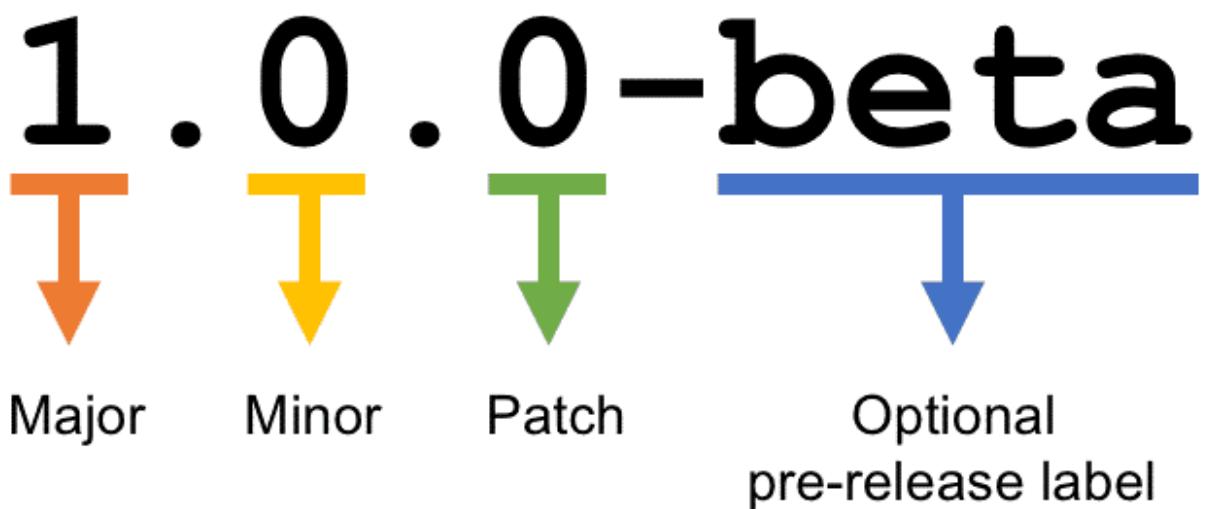
## Diario de experiencias de laboratorio

---



# Diario de experiencias del laboratorio

Cada práctica de laboratorio deberá publicarse en una carpeta en el diario de experiencias de laboratorio, bajo la versión 1.0.0.



# Referencias

---

## Referencias

Bordigón, F. e Iglesias, A. (2020). Introducción al pensamiento computacional. Universidad

Pedagógica Nacional, Educar Sociedad del Estado y el Ministerio de Educación Argentina.

Beecher, K. (2017). Computational Thinking. A beginner's guide to problem-solving and programming.

ISTE (2018). Computational Thinking. Meets Students Learning. International Society for Technology in Education.

Microsoft (2022). Documentación de C#. <https://docs.microsoft.com/es-es/dotnet/csharp/>

MIT (2022). Computational Thinking, a live online Julia/Pluto textbook. Julia: A Fresh Approach to Computing. [computationalthinking.mit.edu](http://computationalthinking.mit.edu). Massachusetts Institute of Technology

Pourbahrami & Tritty (2018). Computational Thinking: How Computer Science Is Revolutionizing Science and Engineering. ENGenious (15) 8-11. <https://resolver.caltech.edu/CaltechCampusPubs:20181025-110029157>.

## Créditos

---

