

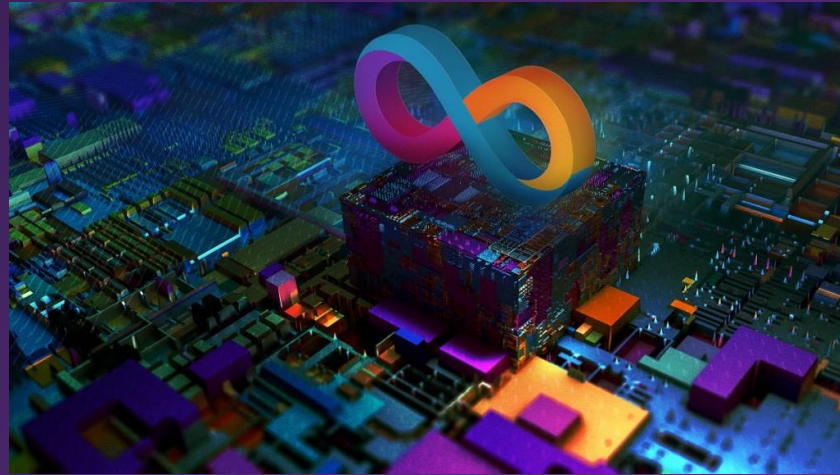
# Crea tu primera dApp con ICP



# Noción — general del curso

En este curso, vamos a crear una dApp (aplicación descentralizada) utilizando Motoko y la infraestructura de Internet Computer.

Vamos a crear nuevos tokens.



# ¿Quién está hablando?

Consultor en  
muchas  
tecnologías.

Crypto-  
inversionista

Líder de  
comunidades  
técnicas.

Emprendedor en  
tecnologías Blockchain



## — CONTENIDOS

- ¿Cómo funciona la infraestructura de ICP?
- ¿Qué es Motoko?
- ¿Qué son los tokens?
- Creando los contratos con Motoko usando Playground.
- Desplegar los contratos con DFINITY Canister SDK.



# Blockchain e Internet \_ Computer



## — Curso introductorio

### Youtube

<https://youtube.com/playlist?list=PLu4f2kXcjVZYJoDKTzgfuRu35zn13bxf>



### Presentación

[https://github.com/web3-explorers/icp\\_introinternetcomputer](https://github.com/web3-explorers/icp_introinternetcomputer)



# ¿Cómo funciona la — infraestructura de ICP?



# ¿De qué — vamos a hablar?

Vamos a ver los principales componentes a tener en cuenta al desarrollar con ICP.





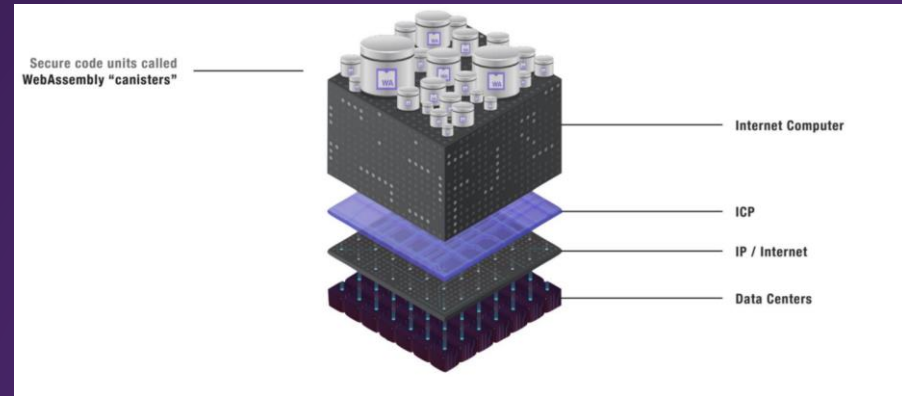
## — Canisters

Es la evolución de los Smart Contracts.

Almacenan el programa y su estado.

Permiten desarrollar todo tipo de proyectos

Se requiere saber Rust o Motoko.



## — Ciclos

Es el “activo” utilizado para pagar por los recursos que se consumen en la red.

Hay que convertir desde ICP a Cycles.



## — Identidad

<https://identity.ic0.app/>



### Welcome to Internet Identity

Provide an Identity Anchor to authenticate.

Authenticate

New? **Create an Internet Identity Anchor.**

Already have an anchor **but using a new device?**

Lost access **and want to recover?**

[ABOUT](#) · [FAQ](#)

— ¿Qué es Motoko?



# ¿De qué — vamos a hablar?

Necesitamos  
conocer cuál es el  
lenguaje de  
programación para  
los Canister en ICP,  
Motoko.



# — Motoko

Es el lenguaje de programación nativo para los Canisters.

```
import Text "mo:base/Text";
import Map "mo:base/HashMap";

actor Registry {

  let map = Map.HashMap<Text, Nat>(10, Text.equal, Text.hash);

  public func register(name : Text) : async () {
    switch (map.get(name)) {
      case null {
        map.put(name, map.size());
      };
      case (?id) { };
    }
  };

  public func lookup(name : Text) : async ?Nat {
    map.get(name);
  };
};

await Registry.register("hello");
(await Registry.lookup("hello"), await Registry.lookup("world"))
```

## — ¿De qué se compone?

- Actor
- Objetos locales
- Gestión de errores
- Módulos
- Funciones asíncronas
- Y más.

```
import Text "mo:base/Text";
import Map "mo:base/HashMap";

actor Registry {

  let map = Map.HashMap<Text, Nat>(10, Text.equal, Text.hash);

  public func register(name : Text) : async () {
    switch (map.get(name)) {
      case null {
        map.put(name, map.size());
      };
      case (?id) { };
    }
  };

  public func lookup(name : Text) : async ?Nat {
    map.get(name);
  };
};

await Registry.register("hello");
(await Registry.lookup("hello"), await Registry.lookup("world"))
```

## — Sintaxis de variables inmutables

- `let mi_variable = 1;`
- `let mi_variable : Nat = 1;`
- `let mi_texto : Text = "cadena";`
- `let mi_arreglo = ["hola", "mundo"];`
- `let mi_par (Text, Nat) = ["hola", 1];`





## — Sintaxis de variables mutables

- `var` mi\_variable = 1;
- `var` mi\_variable : Nat = 1;
- `var` mi\_texto : Text = "cadena";

[illegible]

## Objetos (encapsular estado)

```
object counter {  
  var count = 0;  
  public func inc() { count += 1 };  
  public func read() : Nat { count };  
  public func bump() : Nat {  
    inc();  
    read()  
  };  
};
```

[illegible]

## — Clase (encapsular estado)

```
class Counter(){
  var c = 0;
  public func inc(): Nat {
    c += 1;
    return c;
  }
};
```

[illegible]

## — Clase (encapsular estado)

```
class Counter(contador : Nat){
  var c = contador;
  public func inc(): Nat {
    c += 1;
    return c;
  }
};
```

[illegible]

## — Actor (abstracción de un Canister)

```
actor Counter {  
  var count = 0;  
  public func inc() : async () { count += 1 };  
  public func read() : async Nat { count };  
  public func bump() : async Nat {  
    count += 1;  
    count;  
  };  
};
```

[illegible]



## — Importar

```
import Array "mo:base/Array";  
import Result "mo:base/Result";  
import Render "mo:redraw/Render";  
import Types "types";  
import Utils "utils";
```



## ***Exploremos algunos ejemplos con Motoko Playground***

**[https://m7sm4-2iaaa-aaaab-  
qabra-cai.raw.ic0.app/](https://m7sm4-2iaaa-aaaab-qabra-cai.raw.ic0.app/)**

– ¿Qué son los tokens?





# ¿De qué — vamos a hablar?

Es hora de hablar del  
fundamento sobre  
tokens antes de  
programar.



## — Cripto-monedas

Es el principal activo en una Blockchain, se utiliza para las distintas transacciones y comisiones.

Ejemplo:

- ICP
- Ethereum (ETH)
- Bitcoin (BTC)



## — Cripto-tokens

Son activos creados y mantenidos dentro de una Blockchain con un fin particular

Ejemplo:

- Cycles
- ChainLink(LINK)
- UniSwap(UNI)



## — Ventajas

- Capacidad de programar distintas capacidades.
- Enfocados en temas particulares.
- Es la mejor forma de entregar propiedad o parte de un activo físico.



# Creando los contratos — con Motoko



# ¿De qué — vamos a hablar?

Ahora vamos a  
construir las  
funcionalidades, con  
Motoko



## — Github

- Todo el código está disponible en GitHub:

[https://github.com/web3-explorers/icp\\_creandotuprimeradapp](https://github.com/web3-explorers/icp_creandotuprimeradapp)



# Desplegar los contratos — con DFINITY Canister SDK





# ¿De qué — vamos a hablar?

Desplegaremos  
nuestra dApp  
mediante el SDK.



## — Paso 1

➤ Vamos a clonar el código

```
git clone https://github.com/web3-explorers/icp\_creandotuprimeradapp.git
```

## — Paso 2

- Vamos a instalar el Dfinity Canister SDK (necesitas un entorno Linux)

```
sudo sh -ci "$(curl -fsSL https://smartcontracts.org/install.sh)"
```

## — Paso 3

- Vamos a ejecutar el Canister y sus distintas funciones.

**Fin del curso**

## — Repaso

Entendimos la  
infraestructura de ICP

1

Aprendimos sobre tokens

3

Utilizamos Dfinity  
Canister SDK

5

Aprendimos Motoko y la  
descentralización

2

Creamos el código

4

Probamos nuestra dApp

6

## — Mantén el contacto

### Redes sociales

<https://twitter.com/dfinity>

<https://www.linkedin.com/company/dfinity/>


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

<https://t.me/dfinity/>

<https://medium.com/dfinity>

<https://www.facebook.com/dfinity.org>





**Ahora es tu turno  
— de llevar a cabo  
tus ideas con ICP.**