Tipos de datos
- Sintaxis
- Clasificación
  - Escalares
    - Entero
    - Flotante
    - Booleano
    - Carácter
  - Valores nulos
  - Compuestos
    - Tupla
    - Array
  - Complejos

Tipos de datos

- Cada valor en el lenguaje Rust es de cierto tipo de datos, por lo tanto las variables y constantes están asociadas con un tipo de datos.
- El tipo de datos determina:
  - El tamaño y diseño de la memoria de la variable.
  - El rango de valores que se puede almacenar dentro de esa memoria.
  - El conjunto de operaciones que se puede realizar en la variable.
- El **sistema de tipos** representa los diferentes tipos de valores admitidos por el lenguaje y verifica la validez de los valores suministrados antes que el programador los manipule.

Sintaxis

- Con tipo dato no especificado

```
let salario=3_500.00;
println!("su salario es {}",salario);
```

- Con tipo dato especificado

```
let salario:f64=3_500.00;
println!("su salario es {}",salario);
```

Clasificación

- Según la cantidad de valores almacenados y el valor almacenado

Escalares

- Permite que la variable o constante con ese tipo de datos almacene **un solo valor**

Entero

```
let por_defecto=50;
let edad:u8=27;
let resta:i16=9_731-7_834;
let marca:isize=20;
let cantidad: usize=456;

println!("por defecto: {}",por_defecto);
println!("edad: {}",edad);
println!("resta: {}",resta);
println!("marca: {}",marca);
println!("cantidad: {}",cantidad);
```

- Los enteros no tiene componente fraccionario.
- El lenguaje rust utilizar el tipo de datos entero de tamaño **i32** por defecto, funciona rápido en las máquinas con arquitectura de x86 y x64 bits.
- **Enteros con signo (i)**
  - Se almacenan utilizando la representación de complemento a dos.
  - **Limite** `-[2^(n - 1)] a [2^(n - 1)]-1`, donde `n` es el número de bits que la variante use.
- **Enteros sin signo (u)**
  - Permite establecer una mayor cantidad de elementos que con signo.
  - **Limite** `0 a (2^n)-1`, donde `n` es el número de bits que la variante use.
- **Tipos de enteros**

Longitud	Con signo	Sin signo
8 bits	i8	u8
16 bits	i16	u16
32 bits	i32	u32
64 bits	i64	u64
128 bits	i128	u128
arch	isize	usize

- Los tipos de tamaño **isize** y **usize** dependen del tipo de arquitectura de la máquina donde se ejecute y son utilizados para indexar algun tipo de colección.

```
let a:u8=256;   // 256 desborda en 1
let b:i8=128;   // 129 desborda en 2
println!("numeros: {},{}",a,b);
```

- **Overflow**, el desbordamiento ocurre cuando se sobrepasa la capacidad del entero.

Flotante

```rust
let resultado=10.01;
let interes:f32=8.35;
let costo:f64=12_000.785;

println!("resultado: {}",resultado);
println!("interes: {}",interes);
println!("costo: {}",costo);
```

- Los flotantes tienen componente fraccionario.
- Están en base a la **IEEE-754**.
- El lenguaje rust utilizar el tipo de datos flotante de tamaño **f64** por defecto, alcanza la misma velocidad que **f32**, pero tiene mayor precisión.
- Para facilitar la lectura se puede utilizar el `_` guión bajo entre los números.
- **Tipos de flotante**

Longitud	Tipos	Precisión
32 bits	f32	Simple
64 bits	f64	Doble

Booleano

```rust
let esta_jugando:bool=false;
println!("¿esta jugando?: {}",esta_jugando);
```

- Los flotantes tienen dos valores posibles **true** o **false**.
- Utilice la palabra clave **bool** para declarar una variable booleana.
- Su tamaño es de 1 byte = 8 bits = i8

Carácter

```rust
let caracter_especial='@';
let caracter_alfanumerico: char='7';
let caracter_emoji:char = '😁';

println!("caracter especial: {}",caracter_especial);
println!("caracter alfanumérico: {}",caracter_alfanumerico);
println!("caracter emoji: {}",caracter_emoji);
```

- El lenguaje rust utilizar el tipo de datos carácter, que admite alfanuméricos, unicode (puede representar mucho más que solo ASCII) y caracteres especiales.
  - Letras acentuadas
  - Caracteres chinos, japoneses y coreanos
  - Emojis
  - Espacios de ancho cero
- Es el tipo de datos de manipulación de caracteres más primitivo del lenguaje.
- Se utiliza la palabra clave **char** para declarar una variable de tipo carácter.
- El carácter va encerrado entre comillas simples `'valor'`.

Valores nulos

- Para el lenguaje rust, nulo es un valor que actualmente no es válido o está ausente por algún motivo
- En el lenguaje rust **no existen los valores nulos**, esto evita problemas, porque la referencia nula genera errores, vulnerabilidades y fallas del sistema (*Tony Hoare, error de mil millones de dólares*).

Compuestos

- Permite que la variable o constante con ese tipo de datos almacene **más de un solo valor**

Tupla

```
  let tupla = (true,'X','Y','Z',56_854); // Se declara una tupla sin
especificar los tipos de datos
  println!("valores de la tupla {:?}",tupla);

  let tupla:(f64,char,bool)=(789_345.789,'😁',true); // Se declara una tupla
especificando los tipos de datos
  println!("valores de la tupla {:?}",tupla);
  println!("primer valor de la tupla {:?}",tupla.0);
  println!("segundo valor de la tupla {:?}",tupla.1);
  println!("tercer valor de la tupla {:?}",tupla.2);
```

- Los valores almacenados en las tuplas pueden ser de **diferente tipo de datos**.
- Las tuplas **tienen una longitud fija una vez declaradas**, es decir no pueden incrementar o reducir su longitud despues de su declaración.
- El **índice** de la tupla comienza en **cero**.

```
  let tupla = (true,'X','Y','Z',56_854);
  let (a,b,c,d,e) = tupla;

  println!("primer valor de la tupla {:?}",a);
  println!("segundo valor de la tupla {:?}",b);
  println!("tercer valor de la tupla {:?}",c);
  println!("cuarto valor de la tupla {:?}",d);
  println!("quinto valor de la tupla {:?}",e);
```

- Las tuplas permiten su **desestructuración**, es decir se desempaquetan los valores de una tupla en distintas variables.

Array

```
  let array =['A','B','C']; // Se declara e inicializa un array sin
especificar sus tipos de datos
  println!("valores del array: {:?}",array);
  let array:[i32;5]=[1,2,3,4,5]; // Se declara e inicializa un array
especificando sus tipos de datos
  println!("valores del array: {:?}",array);
  let array=[-2.5;10]; //Se declara e inicializa un array sin especificar sus
tipos de datos y con valores predeterminados de -2.5
  println!("valores del array: {:?}",array);
  println!("tamaño del array: {}",array.len());
  let array:[f64;10]=[-2.5;10]; //Se declara e inicializa un array
especificando sus tipos de datos y con valores predeterminados de -2.5
  println!("valores del array: {:?}",array);
  println!("tamaño del array: {}",array.len());
```

- Los valores almacenados en los arrays deben ser del **mismo tipo de datos**.
- Los arrays **tienen una longitud fija una vez declarados**, es decir no pueden incrementar o reducir su longitud despues de su declaración.
- La declaración de un array asigna bloques de memoria secuenciales y cada bloque de memoria representa un elemento del array.
- Completar los valores de una array se le conoce como inicialización del array.

```
    let mut meses = ["Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio",
"Julio", "Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre"];
    println!("meses del año: {:?}", meses);
    println!("mes 6: {:?}", meses[5]);
    meses[5]="Marzo"; // Se actualiza el valor con el indice 5 del array
    println!("mes 6 (modificado): {:?}", meses[5]);// Se consulta el valor
modificado
```

- Los valores de un array se identifican mediante un entero único llamado **índice**.
- El lenguaje rust entra en **pánico** si intenta acceder por el **indice** a un valor no válido o no existente.
- Los valores del array se pueden **consultar, insertar, modificar pero no eliminar**.

Complejos

- Falta...