Rust

➢ Variables:

Use **let** to declare variables:

Example:

```
let company_string = "TutorialsPoint";  // string type
   let rating_float = 4.5;                 // float type
   let is_growing_boolean = true;          // boolean type
   let icon_char = '♥';                  //unicode character
type
```

```
let variable_name = value;            // no type specified
let variable_name:dataType = value;   //type specified
```

example:

```
let fees = 25_000;
let salary:f64 = 35_000.00;
```

➢ Variables in rust are immutable:

```
let fees = 25_000;

   fees = 35_000;
```

This will throw an error as follows:

```
error[E0384]: re-assignment of immutable variable `fees`
 --> main.rs:6:3
   |
 3 | let fees = 25_000;
   | ---- first assignment to `fees`
...
 6 | fees=35_000;
   | ^^^^^^^^^^^ re-assignment of immutable variable
```

➢ To declare a mutable variable:

```
let mut variable_name = value;
let mut variable_name:dataType = value;
```

example:

```
let mut fees:i32 = 25_000;
fees = 35_000;
```

- **Declaring constants:**

```
const VARIABLE_NAME:dataType = value;
```

Example:

```
   const USER_LIMIT:i32 = 100;     // Declare a integer
constant
   const PI:f32 = 3.14;            //Declare a float constant
```

- **Declaring Strings**

  String literal:

  A simple variable containing a string can be declared as:

```
let location:&str = "Hyderabad";
```

String Object:

Creating a empty string object:

```
    String::new()
```

Example:

```
let empty_string = String::new();
```

Creating a string object from a string:

```
    String::from()
```

Example:

```
let content_string = String::from("String value");
```

String object methods:

Sr.No.	Method	Signature	Description
1	new()	pub const fn new() → String	Creates a new empty String.
2	to_string()	fn to_string(&self) → String	Converts the given value to a String.

3	replace()	pub fn replace<'a, P>(&'a self, from: P, to: &str) → String	Replaces all matches of a pattern with another string.
4	as_str()	pub fn as_str(&self) → &str	Extracts a string slice containing the entire string.
5	push()	pub fn push(&mut self, ch: char)	Appends the given char to the end of this String.
6	push_str()	pub fn push_str(&mut self, string: &str)	Appends a given string slice onto the end of this String.
7	len()	pub fn len(&self) → usize	Returns the length of this String, in bytes.
8	trim()	pub fn trim(&self) → &str	Returns a string slice with leading and trailing whitespace removed.
9	split_whitespace()	pub fn split_whitespace(&self) → SplitWhitespace	Splits a string slice by whitespace and returns an iterator.
10	split()	pub fn split<'a, P>(&'a self, pat: P) → Split<'a, P> , where P is pattern can be &str, char, or a closure that determines the split.	Returns an iterator over substrings of this string slice, separated by characters matched by a pattern.

11	chars()	pub fn chars(&self) → Chars	Returns an iterator over the chars of a string slice.

➢ Decision making

IF Statement:

```
let num:i32 = 5;
   if num > 0 {
      println!("number is positive") ;
   }
```

IF ELSE:

```
let num = 12;
   if num % 2==0 {
      println!("Even");
   } else {
      println!("Odd");
   }
```

MATCH Statement: The match statement checks if a current value is matching from a list of values, this is very much similar to the switch statement in C language.

```
   let state_code = "MH";
   let state = match state_code {
      "MH" => {println!("Found match for MH"); "Maharashtra"},
      "KL" => "Kerala",
      "KA" => "Karnadaka",
      "GA" => "Goa",
      _ => "Unknown"
   };
```

➢ LOOPS:

For loop:

Syntax:

```
for temp_variable in lower_bound..upper_bound {
   //statements
}
```

Example:

```
for x in 1..11{ // 11 is not inclusive
      if x==5 {
         continue;
      }
      println!("x is {}",x);
   }
```

While Loop:

Example:

```
while x < 10{
      x+=1;
      println!("inside loop x value is {}",x);
   }
```

Loop loop:

This loops runs continuously with a break statement is encountered.

Example:

```
   let mut x = 0;
   loop {
      x+=1;
      println!("x={}",x);

      if x==15 {
         break;
      }
   }
```

Continue statement:

The continue statement skips the subsequent statements in the current iteration and takes the control back to the beginning of the loop.

Example:

```
for num in 0..21 {
    if num % 2==0 {
        continue;
    }
    count+=1;
}
```

➢ Functions:

Declaring a function:

Syntax

```
fn function_name(param1,param2..paramN) {
    // function body
}
```

Example

```
fn fn_hello(){
   println!("hello from function fn_hello ");
}
```

Calling a function:

Syntax:

```
    function_name(val1,val2,valN)
```

Example:

```
    fn_hello();
```

Function with return statement:

Syntax:

```
fn function_name() -> return_type {
   //statements
   return value;
}
```

Syntax 2: without return statement:

```
//Syntax2
fn function_name() -> return_type {
   value //no semicolon means this value is returned
}
```

Example:

```
fn main(){
   println!("pi value is {}",get_pi());
}
fn get_pi()->f64 {
   22.0/7.0
}
```

Function pass by value:

When a method is invoked, a new storage location is created for each value parameter. The values of the actual parameters are copied into them. Hence, the changes made to the parameter inside the invoked method have no effect on the argument.

```
fn main(){
   let no:i32 = 5;
   mutate_no_to_zero(no);
   println!("The value of no is:{}",no);
}

fn mutate_no_to_zero(mut param_no: i32) {
   param_no = param_no*0;
   println!("param_no value is :{}",param_no);
}
```

Function pass by reference:

When you pass parameters by reference, unlike value parameters, a new storage location is not created for these parameters. The reference parameters represent the same memory location as the actual parameters that are supplied to the method. Parameter values can be passed by reference by prefixing the variable name with an **&**

```
fn main() {
   let mut no:i32 = 5;
   mutate_no_to_zero(&mut no);
   println!("The value of no is:{}",no);
}
fn mutate_no_to_zero(param_no:&mut i32){
   *param_no = 0; //de reference
}
```

Passing string to function

```
fn main(){
   let name:String = String::from("TutorialsPoint");
   display(name);
   //cannot access name after display
}
fn display(param_name:String){
   println!("param_name value is :{}",param_name);
}
```

➢ Arrays:

Declaring arrays

```
//Syntax1
let variable_name = [value1,value2,value3];

//Syntax2
let variable_name:[dataType;size] = [value1,value2,value3];

//Syntax3
let variable_name:[dataType;size] =
[default_value_for_elements,size];
```

Simple array:

```
let arr:[i32;4] = [10,20,30,40];
```

Without data type:

```
let arr = [10,20,30,40];
```

Array with for loop:

```
   let arr:[i32;4] = [10,20,30,40];

   for index in 0..4 {
      println!("index is: {} & value is :
{}",index,arr[index]);
   }
```

➢ More references:

https://www.javatpoint.com/rust-tutorial
https://www.geeksforgeeks.org/rust-basics/
https://www.tutorialspoint.com/rust/rust_tutorial.pdf
https://www.tutorialspoint.com/rust/index.htm
https://doc.rust-lang.org/stable/rust-by-example/
https://learning-rust.github.io/docs/a6.variable_bindings,constants_and_statics.html