```
# Genera 5 números aleatorios entre 0 y 1
numeros <- runif(5, min = 0, max = 1)
print(numeros)

num <- 3.14
texto <- "R"
print(class(num))    # Imprime "numeric"
print(class(texto))  # Imprime "character"

biseccion = function(funcion, extremoA, extremoB) {
 tolerancia = 0.01
 iteracionMaxima = 100

 # Verificar que f(extremoA) y f(extremoB) tengan signos opuestos
 if (funcion(extremoA) * funcion(extremoB) >= 0) {
  stop("La función debe tener signos opuestos en extremoA y extremoB.")
 }

 errorAproximado = NA
 errorPorcentual = NA

 for (iter in 1:iteracionMaxima) {
  puntoMedio = (extremoA + extremoB) / 2

  # Calcular el error aproximado (se usa la mitad del ancho del intervalo)
  errorAproximado = abs(extremoB - extremoA) / 2
  if (abs(puntoMedio) > 0) {
   errorPorcentual = (errorAproximado / abs(puntoMedio)) * 100
  } else {
```

```
    errorPorcentual = errorAproximado * 100
  }

  # Condición de paro: si la función evaluada en el punto medio es menor que la tolerancia
  # o si el error aproximado es menor que la tolerancia
  if (abs(funcion(puntoMedio)) < tolerancia || errorAproximado < tolerancia) {
    return(list(raiz = puntoMedio, iteraciones = iter,
                errorAproximado = errorAproximado,
                errorPorcentual = errorPorcentual))
  }

  # Actualización del intervalo
  if (funcion(extremoA) * funcion(puntoMedio) < 0) {
    extremoB = puntoMedio
  } else {
    extremoA = puntoMedio
  }
 }
 return(list(raiz = puntoMedio, iteraciones = iteracionMaxima,
             errorAproximado = errorAproximado,
             errorPorcentual = errorPorcentual))
}
```

```
falsa_posicion = function(funcion, extremoA, extremoB) {
 tolerancia = 0.01
 iteracionMaxima = 100

 if (funcion(extremoA) * funcion(extremoB) >= 0) {
   stop("La función debe tener signos opuestos en extremoA y extremoB.")
```

```
  }

  errorAproximado = NA
  errorPorcentual = NA
  aproximacionAnterior = NA

  for (iter in 1:iteracionMaxima) {
    puntoC = (extremoA * funcion(extremoB) - extremoB * funcion(extremoA)) /
(funcion(extremoB) - funcion(extremoA))

    # Calcular error solo si ya hay un valor anterior
    if (!is.na(aproximacionAnterior)) {
      errorAproximado = abs(puntoC - aproximacionAnterior)
      if (abs(puntoC) > 0) {
        errorPorcentual = (errorAproximado / abs(puntoC)) * 100
      } else {
        errorPorcentual = errorAproximado * 100
      }
    }

    if (abs(funcion(puntoC)) < tolerancia || (!is.na(errorAproximado) && errorAproximado <
tolerancia)) {
      return(list(raiz = puntoC, iteraciones = iter,
                  errorAproximado = errorAproximado,
                  errorPorcentual = errorPorcentual))
    }

    if (funcion(extremoA) * funcion(puntoC) < 0) {
      extremoB = puntoC
    } else {
```

```
    extremoA = puntoC
   }
   aproximacionAnterior = puntoC
  }
  return(list(raiz = puntoC, iteraciones = iteracionMaxima,
              errorAproximado = errorAproximado,
              errorPorcentual = errorPorcentual))
}
```

```
secante = function(funcion, valorInicial0, valorInicial1) {
  tolerancia = 0.01
  iteracionMaxima = 100

  errorAproximado = NA
  errorPorcentual = NA

  for (iter in 1:iteracionMaxima) {
    difFunc = funcion(valorInicial1) - funcion(valorInicial0)
    if (abs(difFunc) < .Machine$double.eps) {
      stop("División por cero o diferencia muy pequeña en la función.")
    }

    nuevoValor = valorInicial1 - funcion(valorInicial1) * ((valorInicial1 - valorInicial0) / difFunc)

    errorAproximado = abs(nuevoValor - valorInicial1)
    if (abs(nuevoValor) > 0) {
      errorPorcentual = (errorAproximado / abs(nuevoValor)) * 100
    } else {
      errorPorcentual = errorAproximado * 100
```

```
    }

    if (errorAproximado < tolerancia) {
      return(list(raiz = nuevoValor, iteraciones = iter,
                  errorAproximado = errorAproximado,
                  errorPorcentual = errorPorcentual))
    }

    valorInicial0 = valorInicial1
    valorInicial1 = nuevoValor
  }
  return(list(raiz = nuevoValor, iteraciones = iteracionMaxima,
              errorAproximado = errorAproximado,
              errorPorcentual = errorPorcentual))
}
```

```
newton = function(funcion, derivada, valorInicial) {
  tolerancia = 0.01
  iteracionMaxima = 100

  errorAproximado = NA
  errorPorcentual = NA

  for (iter in 1:iteracionMaxima) {
    valorDerivada = derivada(valorInicial)
    if (abs(valorDerivada) < .Machine$double.eps) {
      stop("La derivada es cero; no se puede continuar.")
    }
```

```
  nuevoValor = valorInicial - funcion(valorInicial) / valorDerivada
  errorAproximado = abs(nuevoValor - valorInicial)

  if (abs(nuevoValor) > 0) {
    errorPorcentual = (errorAproximado / abs(nuevoValor)) * 100
  } else {
    errorPorcentual = errorAproximado * 100
  }

  if (errorAproximado < tolerancia) {
    return(list(raiz = nuevoValor, iteraciones = iter,
                errorAproximado = errorAproximado,
                errorPorcentual = errorPorcentual))
  }
  valorInicial = nuevoValor
 }
 return(list(raiz = nuevoValor, iteraciones = iteracionMaxima,
             errorAproximado = errorAproximado,
             errorPorcentual = errorPorcentual))
}
```

```
raices_multiples = function(funcion, derivada, segundaDerivada, valorInicial) {
 tolerancia = 0.01
 iteracionMaxima = 100

 errorAproximado = NA
 errorPorcentual = NA

 for (iter in 1:iteracionMaxima) {
```

```
  valorDerivada = derivada(valorInicial)
  valorSegundaDerivada = segundaDerivada(valorInicial)

  denominador = valorDerivada^2 - funcion(valorInicial) * valorSegundaDerivada
  if (abs(denominador) < .Machine$double.eps) {
   stop("Denominador muy pequeño; no se puede continuar.")
  }

  nuevoValor = valorInicial - (funcion(valorInicial) * valorDerivada) / denominador
  errorAproximado = abs(nuevoValor - valorInicial)

  if (abs(nuevoValor) > 0) {
   errorPorcentual = (errorAproximado / abs(nuevoValor)) * 100
  } else {
   errorPorcentual = errorAproximado * 100
  }

  if (errorAproximado < tolerancia) {
   return(list(raiz = nuevoValor, iteraciones = iter,
               errorAproximado = errorAproximado,
               errorPorcentual = errorPorcentual))
  }
  valorInicial = nuevoValor
 }
 return(list(raiz = nuevoValor, iteraciones = iteracionMaxima,
             errorAproximado = errorAproximado,
             errorPorcentual = errorPorcentual))
}
```

```
taylor_exp = function(x, numTerminos) {
  suma = 0
  errorAproximado = NA
  errorPorcentual = NA

  for (n in 0:(numTerminos - 1)) {
    termino = x^n / factorial(n)
    sumaAnterior = suma
    suma = suma + termino

    if (n > 0) {
      errorAproximado = abs(suma - sumaAnterior)
      if (abs(suma) > 0) {
        errorPorcentual = (errorAproximado / abs(suma)) * 100
      } else {
        errorPorcentual = errorAproximado * 100
      }
    }
  }

  return(list(valorAproximado = suma,
              errorAproximado = errorAproximado,
              errorPorcentual = errorPorcentual))
}
```

```
taylor_cos = function(x, numTerminos) {
```

```
  suma = 0
  errorAproximado = NA
  errorPorcentual = NA

  for (n in 0:(numTerminos - 1)) {
    termino = ((-1)^n * x^(2*n)) / factorial(2*n)
    sumaAnterior = suma
    suma = suma + termino

    if (n > 0) {
      errorAproximado = abs(suma - sumaAnterior)
      if (abs(suma) > 0) {
        errorPorcentual = (errorAproximado / abs(suma)) * 100
      } else {
        errorPorcentual = errorAproximado * 100
      }
    }
  }

  return(list(valorAproximado = suma,
              errorAproximado = errorAproximado,
              errorPorcentual = errorPorcentual))
}
```

```
taylor_cos = function(x, numTerminos) {
  suma = 0
  errorAproximado = NA
  errorPorcentual = NA
```

```
  for (n in 0:(numTerminos - 1)) {
    termino = ((-1)^n * x^(2*n)) / factorial(2*n)
    sumaAnterior = suma
    suma = suma + termino

    if (n > 0) {
      errorAproximado = abs(suma - sumaAnterior)
      if (abs(suma) > 0) {
        errorPorcentual = (errorAproximado / abs(suma)) * 100
      } else {
        errorPorcentual = errorAproximado * 100
      }
    }
  }

  return(list(valorAproximado = suma,
              errorAproximado = errorAproximado,
              errorPorcentual = errorPorcentual))
}
```

```
# Definición de la función y sus derivadas
f = function(x) x^2 - 4
df = function(x) 2 * x
ddf = function(x) 2

# Uso de los métodos para encontrar raíces:
resultadoBiseccion = biseccion(f, 0, 3)
cat("Bisección:\n")
print(resultadoBiseccion)
```

```
resultadoFalsaPosicion = falsa_posicion(f, 0, 3)
cat("Falsa Posición:\n")
print(resultadoFalsaPosicion)

resultadoSecante = secante(f, 0, 3)
cat("Secante:\n")
print(resultadoSecante)

resultadoNewton = newton(f, df, 3)
cat("Newton-Raphson:\n")
print(resultadoNewton)

resultadoRaicesMultiples = raices_multiples(f, df, ddf, 3)
cat("Raíces Múltiples:\n")
print(resultadoRaicesMultiples)

# Uso de las series de Taylor:
xValor = 0.5
numTerminos = 10

resultadoExp = taylor_exp(xValor, numTerminos)
cat("Taylor para e^x:\n")
print(resultadoExp)

resultadoSin = taylor_sin(xValor, numTerminos)
cat("Taylor para sin(x):\n")
print(resultadoSin)

resultadoCos = taylor_cos(xValor, numTerminos)
```

```
cat("Taylor para cos(x):\n")
print(resultadoCos)
```