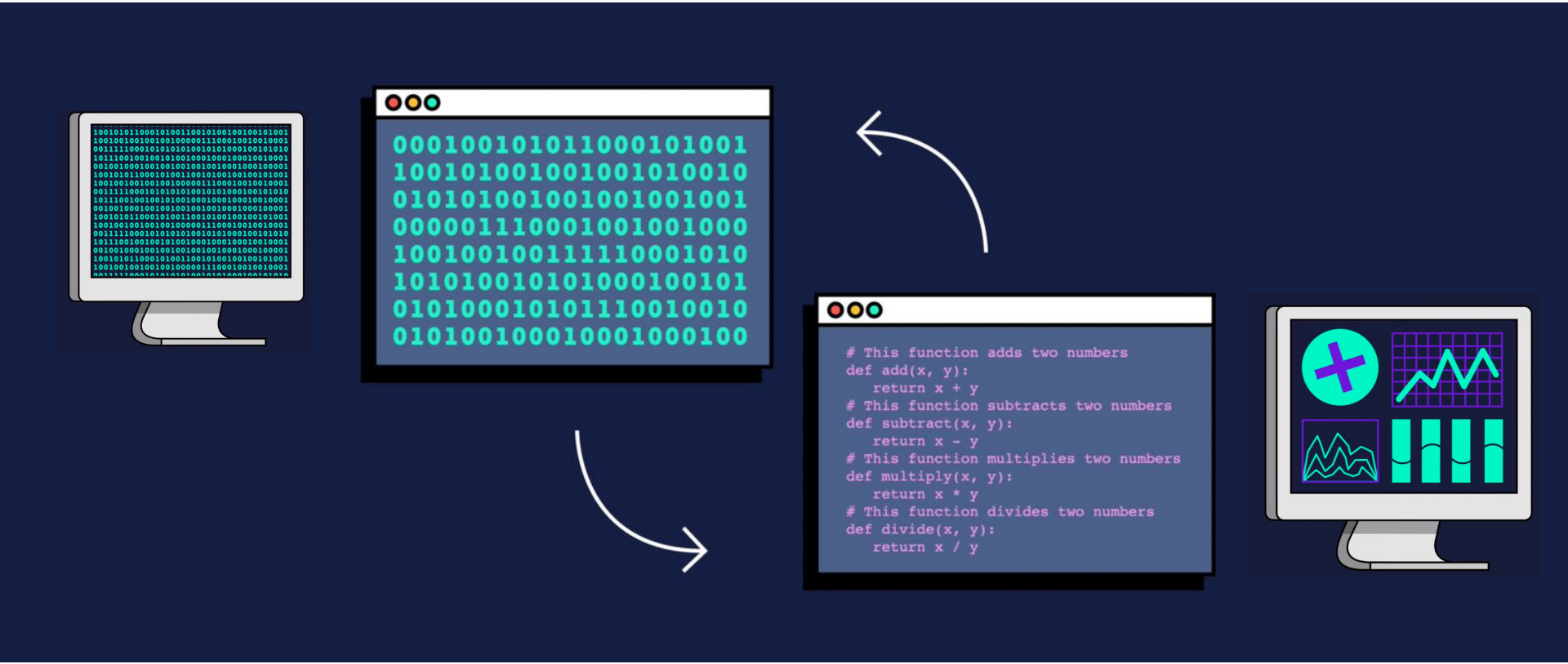
Introducción a la programación en **FORTRAN**

Programación - Lenguaje de programación



Lenguajes de programación

```
print('Hello, world!');
```

JavaScript, Go

Python, Ruby

Java, C#, Visual Basic .NET

C++

```
program hello
    write(*,*) 'Hello, World!'
end program hello
```

Fortran, COBOL

C

```
bdos    equ    0005H
start:  mvi    c,9
        lxi    d,msg$
        call   bdos
        ret
msg$:   db     'Hello, world!$'
end     start
```

Lenguaje ensamblador
(Assembler)

```
48 6F 6C 61 2C
01101000011011110110110001100001
```

Código Máquina
72 6C 4F 9E 21 (Hexadecimal)
01001100101010 (Binario)

CPU (Procesador)

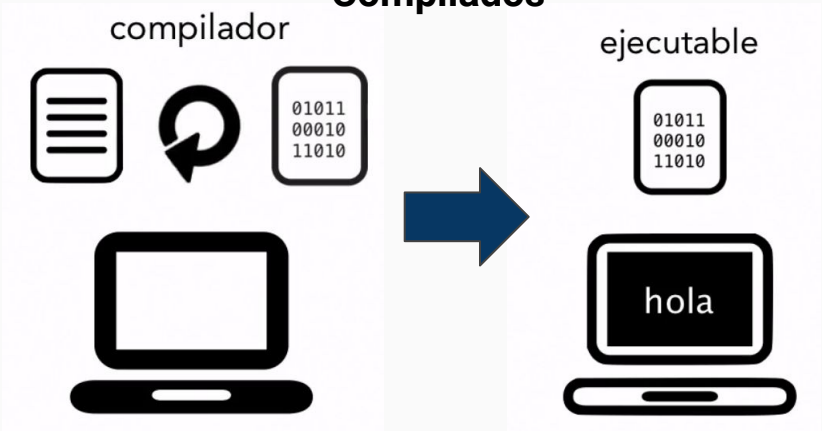
High Level
(Lenguajes de alto nivel)

Low Level
(Lenguajes de bajo nivel)

https://en.wikibooks.org/wiki/Computer_Programming/Hello_world

Tipos de Lenguajes: Compilados - Interpretados

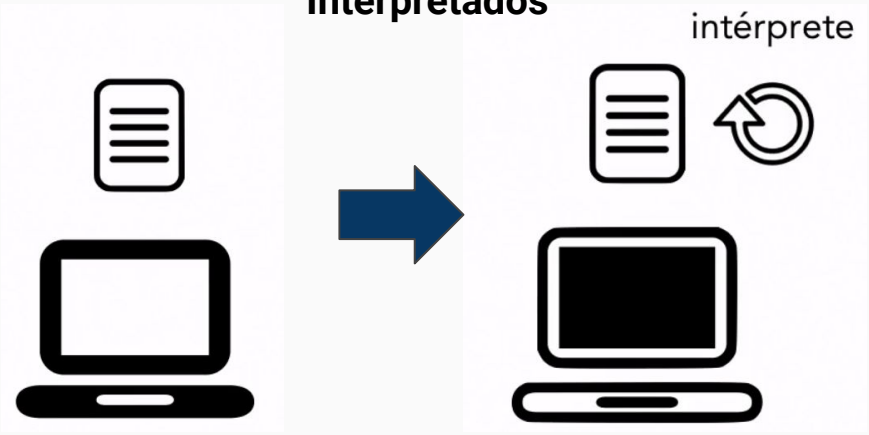
Compilados



COMPILADOS	
preparados para ejecutarse	no son multiplataforma
usualmente más rápidos	poco flexibles
el código fuente es inaccesible	se requiere un paso extra (compilar)

Fortran, C++, C, Go, ...

Interpretados



INTERPRETADOS	
son multiplataforma	se requiere un intérprete
son más sencillos de probar	usualmente más lentos
fácil debugging	el código fuente es público

Java, C#, ...

Ruby, Python, R, JavaScript, ...

FORmula **TRAN**slation System (Fortran)

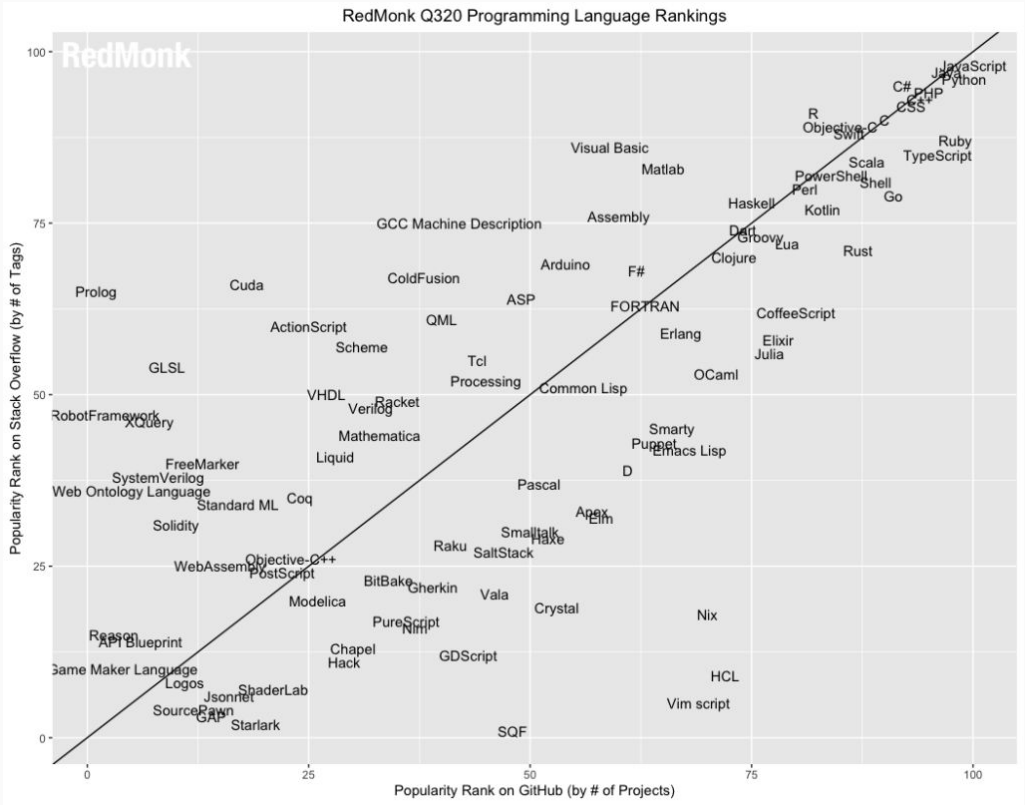
- Primer lenguaje de programación de alto nivel.
- Enfocado al cálculo numérico y a la programación científica.
- Uso continuo en áreas de cálculo intensivo (numerical weather prediction, finite element analysis, computational fluid dynamics, geophysics, computational physics, crystallography and computational chemistry)
- Popular en la computación de alto rendimiento.
- Constante mejora y actualización (versiones)
  - **Fortran 77 :** Soporte para procesamiento de datos basados en caracteres.
  - **Fortran 90 :** Programación de arreglos y programación modular.
  - **Fortran 95 :** Programación de alto desempeño.
  - **Fortran 2003 :** Programación orientada a objetos.
  - **Fortran 2008 :** Programación concurrente.
  - **Fortran 2018 :** Programación en paralelo.

`.f90` `.F95` `.f03`
`.f` `.for` `.F`
`.F90` `.F95` `.F77`

Compiladores

- **GNU Fortran (gfortran)**
  (https://gcc.gnu.org/wiki/GFortran/News#GCC9)
- G95 (descontinuado)
- Intel fortran (ifort)
- NAG
- IBM
- AMD
- OnlineGDB
  (https://www.onlinegdb.com/online_fortran_compiler)
- ...

(https://fortran-lang.org/compilers/)

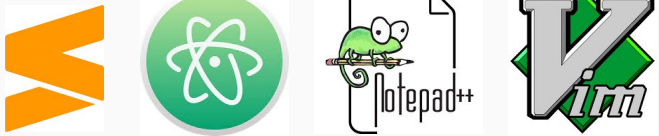


(https://www.tiobe.com/tiobe-index/)

Editores de Código

- No programa de texto enriquecido.
- Resaltado de código.
- Soporta muchos lenguajes.
- Enfocado a archivos.
- Plugins.
- Programación más cómoda y eficiente
- Ligeros

- **Visual Studio Code**
  (https://code.visualstudio.com)
- Sublime
- Atom
- Notepad++
- Vim
- ...




```
test > Ftn var.f90
1   program main
2       real(8) :: a
3       a = 8
4       write(*,*) a
5   end program main
```

main.f90: Bloc de notas

Archivo Edición Formato Ver Ayuda

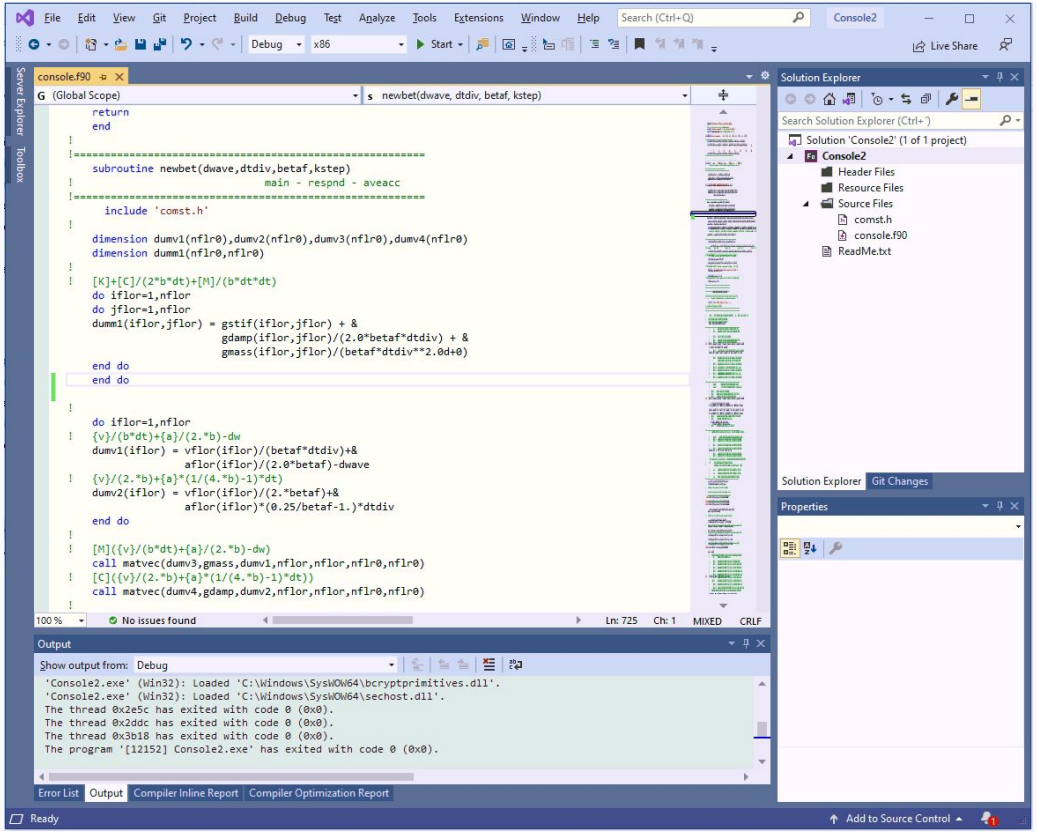
```
program main
    real(8) :: a
    a = 8
    write(*,*) a
end program main
```

100% Windows (CRLF) UTF-8

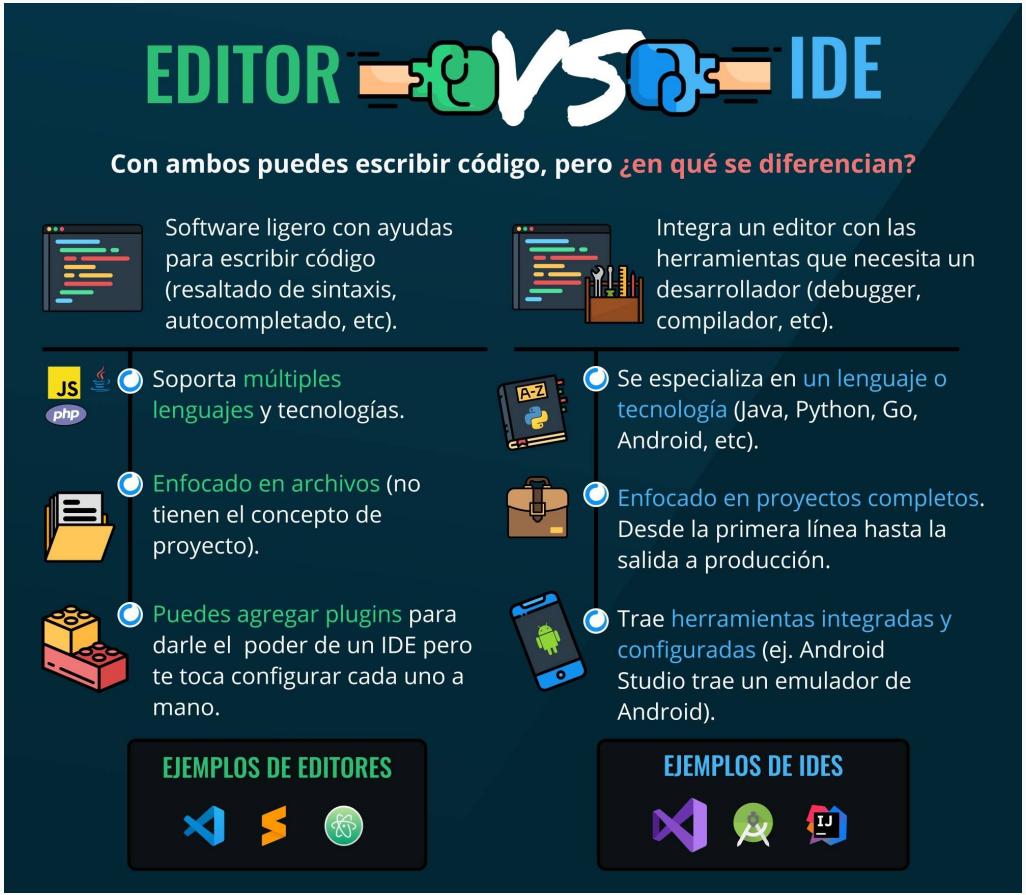
Entorno de desarrollo integrado (IDE)

- Todas las herramientas integradas.
- Estructurado como proyecto.
- Fácil Debugging.
- Integra editor de código y compilador.
- Incluye librerías.
- Más consumo de recursos.

- **Visual Studio IDE** (IDE para C, C++, **Fortran**, Visual Basic, Python, JS, C#, etc) (https://visualstudio.microsoft.com)
- Dev-C++ (IDE para C++)
- Code::Blocks (IDE para C, C++ y **Fortran**) (http://www.codeblocks.org)
- NetBeans (IDE para Java, JS, HTML5, PHP, C++, etc.)
- PyCharm (IDE para Python)
- ...

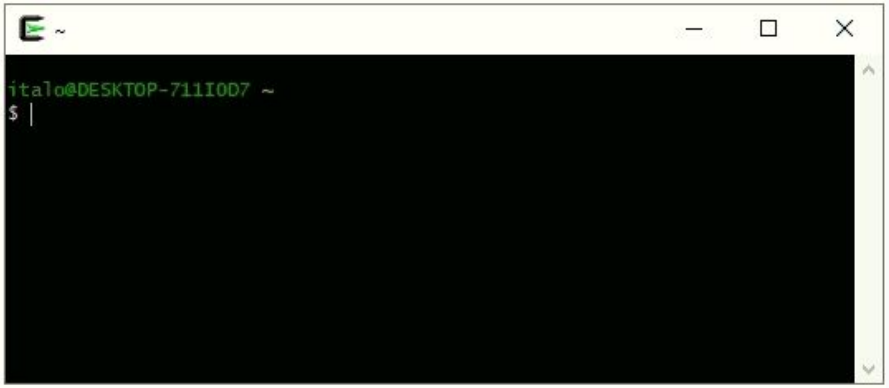


Editor vs. IDE

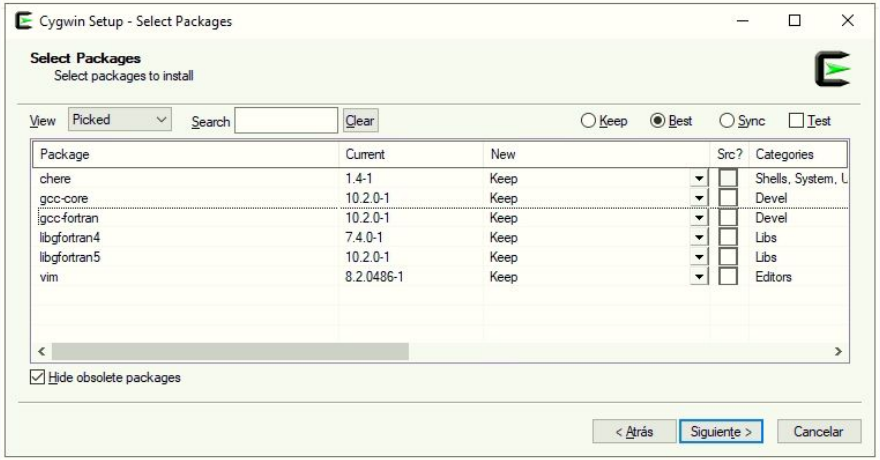


Cygwin

- Colección de herramientas GNU y open source que se ejecutan de manera similar a un sistema operativo Linux en Windows.
  (https://www.cygwin.com)

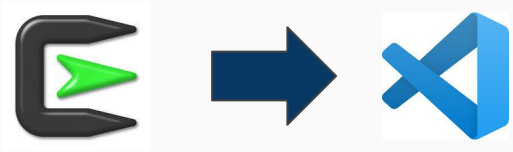


- Descarga (https://www.cygwin.com/setup-x86_64.exe)
- gcc-fortran (compilador gfortran)
- chere (bash cygwin) chere -i -t mintty -s bash
- vim (editor de código)



Cygwin + Visual Studio Code

- Incorporar la terminal de cygwin a la interfaz del Visual Studio Code, para facilitar el trabajo de compilacion y ejecucion.

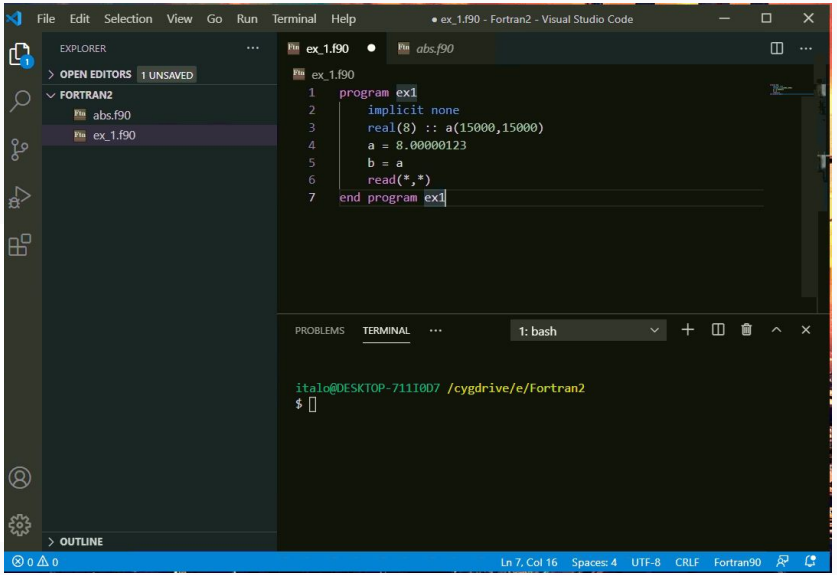


Settings

└ **Terminal Integrated**

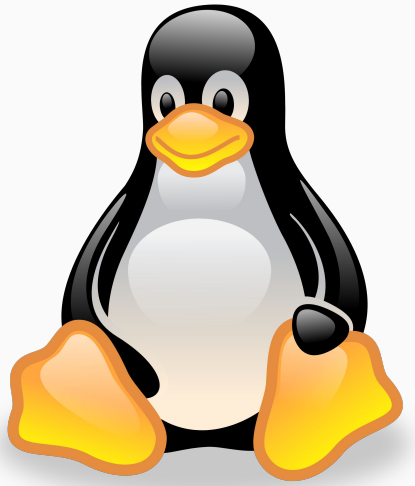
└ **Automation Shell: Windows**

```
{
    "terminal.integrated.shell.windows": "C:\\cygwin64\\bin\\bash.exe",
    "terminal.integrated.env.windows": {

      "CHERE_INVOKING": "1"
    },
    "terminal.integrated.shellArgs.windows": [
      "-l"
    ],

}
```



Comandos Cygwin (Linux OS)

cd namedir	Ingresa al directorio indicado
cd ..	Regresa al directorio superior
ls	Indica la lista de documentos y carpetas
ls -a	Indica la lista e información de documentos y carpetas
mkdir namedir	Crea un nuevo directorio
touch namefile	Crea el archivo indicado
pwd	Indica la dirección de la carpeta actual
rm namefile	Elimina el archivo indicado
rmdir namedir	Elimina la carpeta vacía indicada
rmdir - r namedir	Elimina la carpeta indicada
clear	Borra los comandos ejecutados en la terminal



Comandos gfortran

- **gfortran** file.f90

  Compila el archivo indicado, salida por defecto (a.out o a.exe)
- **gfortran -o** programa.exe file.f90

  Compila el archivo indicado, indicando el nombre del ejecutable
- **./**programa.exe

  Ejecuta el archivo
- **gfortran -o** programa.exe modulo1.f90 modulo2.f90 file.f90

  Compila el archivo indicado y los módulos necesarios
- **gfortran -c** file.f90

  Precompila el archivo, genera un archivo intermedio de extensión **.o**
- **gfortran** file.o

  Termina de compilar el archivo de extensión **.o**
- **gfortran -o** programa.exe modulo1.o modulo2.o file.f90

if … else …

Sintaxis clásica

```
   if (a.eq.b) then
        ! bloque verdad
   else
        ! bloque falso
   end if
```

Sintaxis único bloque

```
   if (a.eq.b) then
        ! bloque verdad
   end if
```

Sintaxis múltiples verificaciones

```
   if (a.eq.b) then
        ! bloque verdad 1
   else if (a.gt.b) then
        ! bloque verdad 2
   else if (a.lt.b) then
        ! bloque verdad 3
   else
        ! bloque falso
   end if
```

Sintaxis única ejecución

```
   if (a.eq.b) write(*,*) a
```

Operaciones Relacionales

- Comparar valores enteros, reales y cadenas de caracteres.

.LT.	<	**L**ess **T**han
.LE.	<=	**L**ess than or **E**qual to
.GT.	>	**G**reater **T**han
.GE.	>=	**G**reater than or **E**qual to
.EQ.	==	**EQ**ual to
.NE.	/=	**N**ot **E**qual to

Operadores Lógicos

- Comparar valores lógicos.

.NOT.	Negación
.AND.	Conjunción
.OR.	Inclusión
.EQV.	Equivalencia
.NEQV.	No equivalencia

```
if (a.eq.b.or.a.gt.c) then
    ! bloque verdad
else
    ! bloque falso
end if
```

select case

- Ejecuta cierto bloque de código según el valor de una variable.
- Generalmente usado con enteros.
- Pero también pueden tomar valores lógicos(**true-false**) y de caracteres.

Sintaxis clásica

```
select case (i)
case (1)
    ! bloque si i es 1
case (2)
    ! bloque si i es 2
case (3)
    ! bloque si i es 3
end select
```

```
select case (i)
case (1)
    ! bloque si i es 1
case (2)
    ! bloque si i es 2
case (3,4,5)
    ! bloque si i es 3,4 o 5
case (6:10)
    ! bloque si i es 6,7,8,9 o 10
case (-2:0)
    ! bloque si i es -2, -1 o 0
case (11:13,18:19)
    ! bloque si i es 11,12,13,18 o 19
case (20:)
    ! bloque si i es mayor o igual a 20
case default
    ! bloque si i toma otro valor
end select
```

do loop

Sintaxis clásica

```
do i = 1,10
   ! bloque de ejecución para cada i
   ! 1,2,3,4,5,6,7,8,9 y 10
end do
```

```
do i = 1,10,2
   ! bloque de ejecución para i :
   ! 1,3,5,7 y 9
end do
```

```
do i = 10,1,-3
   ! bloque de ejecución para i :
   ! 10,7,4,1
end do
```

- Ejecuta cierto bloque de código interno para cada valor asignado.
- i, m, n y j son valores enteros

Sintaxis general

```
do i = m,n,j
   ! bloque de ejecución para i :
   ! desde m hasta n con una variación j
end do
```

do loop: while - cycle - exit

do while

```
do while (i.lt.50)
   ! bloque de ejecución siempre
   ! que se cumpla la condición
end do
```

cycle

```
do i = 1,10
   ! bloque de ejecución 1
   if (j.eq.5) cycle
   ! bloque de ejecución 2
end do
```

exit

```
do ! loop infinito
   ! bloque de ejecución 1
   if (j.eq.5) exit
   ! bloque de ejecución 2
end do
```

- Comandos para romper ciclos o parte de ellos.
- **while:** Verifica una condición para ejecutar
- **cycle:** Termina el ciclo actual de ejecución y pasa al siguiente
- **exit:** Termina todos los ciclos de ejecución

Archivos I/O

- Trabajar las entradas y salidas en archivos caracterizados por unidades.
- Por defecto la unidad 6 es la terminal: **`write(*,*) = write(6,*)`**

```
    open(1, file = 'input.txt', status='old')
    filename = 'num.txt'
    open(2, file = filename, status='new')
    open(i, file = filename, status='unknown')
    ! input
    read(2,*) b
    ! output
    write(1,*) 'el número es:',b


    close(1)
    close(2)
```

Arreglos estáticos

- Se define el tipo de valores del arreglo **(real, entero o complejo)** y las dimensiones de tu arreglo.
- La cantidad de índices/entradas/dimensiones del arreglo se definen separados por comas.
- Por defecto los índices de los arreglos inician en **1.**

```
real(8), dimension(5,5) :: mtx
real(4), dimension(10) :: vec
integer(4), dimension(8,10,15) :: c
real(8) :: mtx2(5,5)
integer(4) :: d
```

- Si se desea iniciar en índice 0 se puede agregar en la definición de la dimensión:

```
real(8) :: mtx2(0:5,0:5)
```

- El llamado y la asignación de valores a ciertos elementos del arreglo se puede realizar de la siguiente forma:

mtx (i , j)

(i,j)	1	2	3	4	5
1	0.5	0.2	0.0	0.0	0.0
2	0.0	0.2	0.0	0.3	0.3
3	0.0	0.2	0.0	0.3	0.3
4	0.0	0.2	0.0	0.3	0.3
5	0.0	0.2	0.0	0.0	0.0

```
mtx = 0.0
mtx(1,1) = 0.5
mtx(:,2) = 0.2
mtx(2:4,4:5) = 0.3
```

Funciones intrínsecas

- **Funciones Numéricas**

  y = **abs**(a)

  m = **max**(a,b,c)

  m = **min**(a,b,c)

  ...

- **Funciones Matemáticas**

  c = **cos**(a) (a en radianes)

  s = **sin**(a)

  t = **tan**(a)

  d = **log**(a)

  d = **log10**(a)

  s = **sqrt**(a)

  ...

- **Multiplicación de Vectores y Matrices**

  y = **dot_product**(vector_a, vector_b)

  Producto escalar entre vectores de la misma dimensión, el resultado es real o entero.

  matrix_p = **matmul**(matrix_a, matrix_b)

  Las matrices deben cumplir las reglas dimensionales de la multiplicación de matrices: matrix_p (n,m), matrix_a(n,p), matrix_b(p,m)

  matrix_t = **transpose**(matrix_a)

  matrix_t (n,m), matrix_a(m,n)

  m = **maxval**(matrix_a)

  m = **minval**(matrix_a)

  ...

(https://docs.oracle.com/cd/E19422-01/819-3684/2_F95_Intrins.html)

Formatos

```
read(*,*)
write( , , )
```

Unidad

Valor numérico entero que indica el lugar **(archivo o terminal)** de donde se leerá o donde se escribirán los valores.

Formato

- **Formato en misma línea**

```
write(*,'(2i10)') a, b
```

- **Formato etiquetado**

```
write(*,100) a, b
100 format(2i10)
```

Números Enteros

Ejemplo imprimir 1432 y 341

2i10.6

				0	0	1	4	3	2					0	0	0	3	4	1

1i7,1i8.7

			1	4	3	2		0	0	0	0	3	4	1

Formatos Números Reales

Números Reales

Ejemplo imprimir c, d : 3.45854612 y 1235.5435734598625431

```
write(*,'(2f10.3)') c,d
```

2f10.3

					3	.	4	5	9			1	2	3	5	.	5	4	4

```
write(*,'(1f7.2,1f8.3)') c,d
```

1f7.2,1f8.3

			3	.	4	6	1	2	3	5	.	5	4	4

```
write(*,'(1f7.2,a,1f8.3)') c,' ',d
```

1f7.2,a,1f8.3

			3	.	4	6		1	2	3	5	.	5	4	4

```
write(*,'(1f7.2,a1,1f7.3)') c,' ',d
```

1f7.2,a,1f7.3

			3	.	4	6		*	*	*	*	*	*	*	*

Formatos Especiales Números Reales

Números Reales

Ejemplo imprimir c, d : 3.45854612 y 1235.5435734598625431

```
write(*,'(2e10.3)') c,d
```

2e10.3

	0	.	3	4	6	E	+	0	1		0	.	1	2	4	E	+	0	4

```
write(*,'(2es10.3)') c,d
```

2e10.3

	3	.	4	5	9	E	+	0	0		1	.	2	3	6	E	+	0	3

```
write(*,'(1f7.2,1e8.1)') c,d
```

1f7.2,1f8.3

			3	.	4	5		0	.	1	E	+	0	4

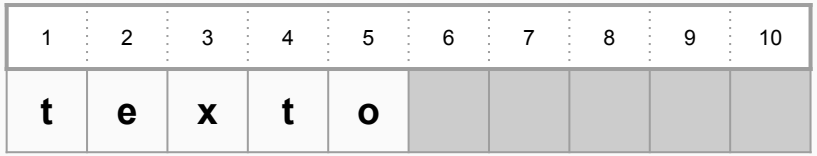
Cadena de caracteres estáticas (strings)

- Se define la cadena de caracteres asignando la dimensión de la cadena.

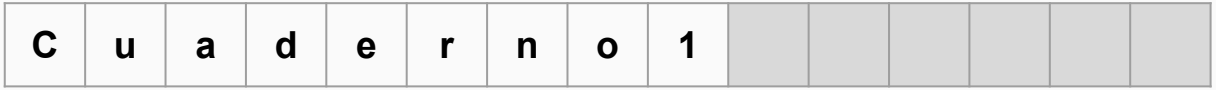
```
character(len=10) :: cad1
character(15) :: cad2
character :: cad3*100
```

- Función **trim(cad),** no considera las espacios nulos de la cadena.
- Las cadenas de caracteres se asignan internamente entre comillas simples o dobles.

```
cad1 = 'texto'
cad2 = 'Cuaderno1'
```




```
cad1(3:4) : 'xt'
cad1(2:2) : 'e'
cad1 : 'texto      '
```



- Concatenación de cadenas **cad1 // cad 2**

```
cad3 = cad1//cad2
```

Funciones

- FORTRAN permite definir funciones al usuario de manera similar a las funciones intrínsecas, con los argumentos de entrada y el resultado de salida que el usuario defina.

Sintaxis de la Función

```
function sqr(x) result(s)
   implicit none
   real, intent(in) :: x
   real :: s
   s = x*x
end function sqr
```

Función en Programa

```
program test
    implicit none
     real :: b,c
     b = 2.0
     c = sqr(b)
     write(*,*) c
end program test
```

- Retorna un único argumento que puede ser un valor o un arreglo de valores.
- Los argumentos de entrada también pueden ser de salida **in out**, y modificarse internamente.
- **return:** nos ayuda a controlar funciones y subrutinas termina la ejecución y retorna los argumentos.

Subroutine

- Bloques de código que realizan operaciones con argumentos de entrada y salida.
- Se utiliza la sentencia **call** para llamar a las subrutinas.

Sintaxis de la Subrutina

```
subroutine sqr2(x,s)
   implicit none
   real, intent(in) :: x
   real, intent(out) :: s
   s = x*x
end subroutine sqr2
```

Subrutina en Programa

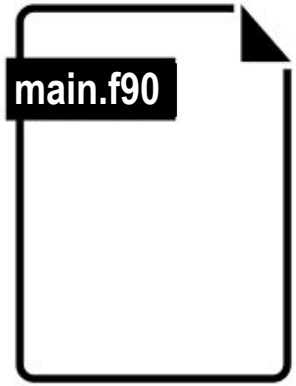
```
program test
     implicit none
     real :: b,c
     b = 2.0
     call sqr2(b,c)
     write(*,*) c
end program test
```

- La subrutina requiere una definición del tipo de cada unas de las variables usadas.
- **Intent:** Permite declarar el tipo de argumento de la subrutina **(in, out, in out)**
- Existen otras opciones de argumentos como el **save, optional, etc.**
- El bloque de código de la subrutina puede encontrarse en el mismo archivo del programa principal, dentro de un **módulo** o se puede agregar con el comando **include**.

(https://en.wikibooks.org/wiki/Fortran/Fortran_procedures_and_functions)

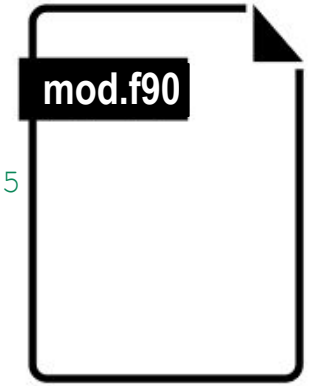
Módulos

- Unidad o bloque de código que contiene constantes, variables, funciones o subrutinas externas al código principal pero que se pueden llamar y utilizar dentro de ella.
- Se compilan junto al programa principal.
- Se llama a todo el módulo dentro del programa principal con la sentencia **USE ...**




```
program test
  use modulo
  implicit none
  real :: b,c
  b = 2.0
  call sqr2(b,c)
  write(*,*) c
end program test
```

```
module modulo
  implicit none
  integer(8) :: i
  real, parameter :: pi = 3.1415
  contains
  subroutine sqr2(x,s)
      implicit none
      real, intent(in) :: x
      real, intent(out) :: s
      s = x*x
  end subroutine sqr2
end module modulo
```



Arreglos Dinámicos

- Permite definir las dimensiones de los arreglos dentro del programa, además de poder modificarse estas dimensiones posteriormente.
- El arreglo se declara con el tipo **allocatable,** en el cual solo se define el rango del arreglo.

```
integer(4), allocatable, dimension(:) :: a
integer(4), allocatable :: b(:)
real(8), allocatable, dimension(:,:) :: c
real(4), allocatable :: d
character(len=:) :: e
```

- Dentro del programa principal se fija la dimensión del arreglo con la declaración **allocate.**

```
allocate(a(10))
allocate(b(15))
n = 12
m = 10
allocate(c(n,m))
allocate(d(0:n))
allocate(character(15) :: e)
```

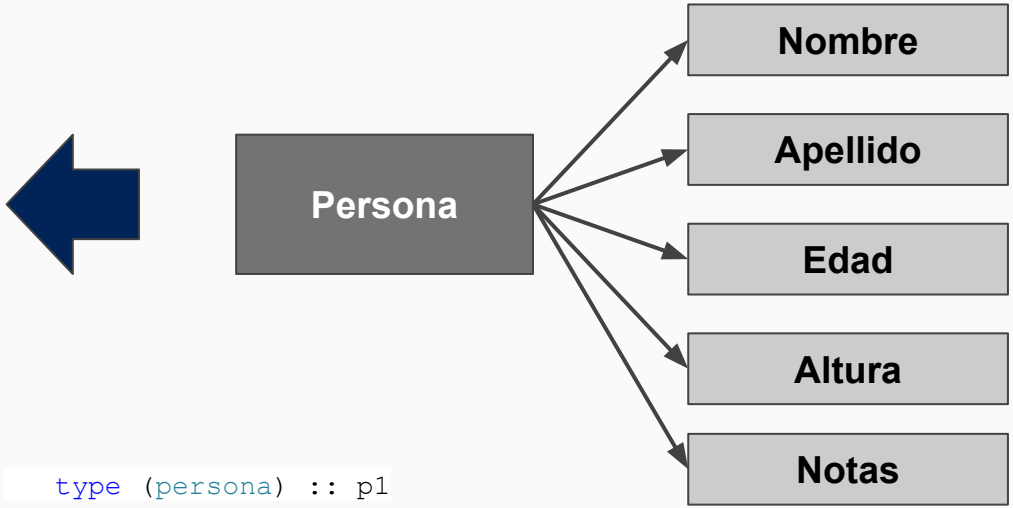
- **deallocate:** Quita la dimensiones definida para el arreglo.

```
deallocate(a)
deallocate(b)
```

Estructuras

- Permite definir una estructura de datos, con argumentos que el usuario requiera.

```
   type persona
        character(len=30) :: nm
        character(len=40) :: ap
        integer :: edad
        real :: alt
        real :: notas(5)
   end type persona
```




```
program str
   implicit none
   type persona
        character(len=30) :: nm
        character(len=40) :: ap
        integer :: edad
        real :: alt
        real :: notas(5)
   end type persona
```

```
   type (persona) :: p1

   p1%nm = 'juan'
   p1%ap = 'Soto'
   p1%edad = 22
   p1%alt = 1.74
   p1%notas(1) = 15.4
   p1%notas(2) = 12.4
```

Código Estructurado