```
! Name of program
! The f90 extension can be used for
! Fortran 2003 code
! Compile like this
! gfortran fortrantut.f90
program fortrantut

  ! A module we will use later compiled
  ! like this
  ! gfortran -c mult_mod.f90  fortrantut.f90
  ! Run like this
  ! gfortran mult_mod.o  fortrantut.o
  ! use mult_mod
  ! use shape
  use shape_mod
  use triangle_mod

  ! Forces you to declare all variables
  implicit none

  ! Create a variable that can hold up to
  ! 20 characters
  character*20 :: name

  ! Print a string to the screen
  ! * use default formatting
  print *, "What's your name : "

  ! Receive input up to a whitespace
  ! or newline
  read *, name

  ! Output the result
  print * , "Hello ", name

  character (len = 20) :: f_name, l_name
  print *, "Enter Name : "
  ! Read 2 values separated by a space
  read *, f_name, l_name
  ! Trim extra whitespace
  print *, "Hello ", trim(f_name), " ", trim(l_name)

  ! ----- VARIABLES / DATA TYPES -----
  ! Variables must start with a letter
  ! and then letters, numbers, _
  ! Variables are case insensitive

  ! Declare a constant that's value
  ! can't change
  real, parameter :: PI = 3.1415

  ! Numbers with decimals (floats)
  ! You can assign a value or leave undefined
  real :: r_num1 = 0.0, r_num2 = 0.0
  ! Doubles are accurate to 15 decimals
  double precision :: dbl_num = 1.1111111111111111d+0
  ! Numbers without decimals (whole numbers)
  integer :: i_num1 = 0, i_num2 = 0
  ! Boolean type
  logical :: can_vote = .true.
  ! Another way to declare a string
  character (len = 10) :: month
  ! Complex TYPES
  complex :: com_num = (2.0, 4.0)

  ! Get largest value for data types
  print *, "Biggest Real ", huge(r_num1)
  print *, "Biggest Int ", huge(i_num1)

```

```
70    ! Get smallest value for data types
71    print *, "Smallest Real ", tiny(r_num1)
72    print *, "Smallest Int ", tiny(i_num1)
73
74    ! Kind returns the number of bytes for each type
75    print "(a4, i1)", "Int ", kind(i_num1)
76    print "(a5, i1)", "Real ", kind(r_num1)
77    print "(a7, i1)", "Double ", kind(dbl_num)
78    print "(a8, i1)", "Logical ", kind(can_vote)
79
80    ! ! ----- FORMATTED OUTPUT WITH PRINT -----
81    ! character(len=5) :: i_char
82    ! Integers are right justified by default
83    print *, "A Number ", 10
84
85    ! Integers are formatted like this RiW
86    ! R : Number of times to use what follows per line
87    ! W : Width to take up for each value
88    print "(3i5)", 7, 6, 8
89    print "(i5)", 7, 6, 8
90
91    ! Floats are formatted like RfW.D
92    ! R & W : Same as above
93    ! D : Decimal places to show
94    print "(2f8.5)", 3.1415, 1.234
95
96    ! Characters & Strings are formatted RaW
97    ! / Adds a newline
98    print "(/, 2a8)", "Name", "Age"
99
100   ! Exponential Notation ReW.D
101   print "(e10.3)",123.456
102
103   ! Use multiple types
104   print "(a5,i2)", "I am ", 43
105
106   ! Left justify Numbers
107   ! Convert int 10 into a string
108   write (i_char, "(i5)") 10
109
110   ! Print formatted output left justified
111   print "(a,a)", "A Number ", adjustl(i_char)
112
113   ! ----- MATH OPERATORS -----
114   real :: float_num = 1.111111111111111
115   real :: float_num2 = 1.111111111111111
116   double precision :: dbl_num = 1.1111111111111111d+0
117   double precision :: dbl_num2 = 1.1111111111111111d+0
118   real :: rand(1)
119   integer :: low = 1, high = 10
120
121   print "(a8,i1)", "5 + 4 = ", (5 + 4)
122   print "(a8,i1)", "5 - 4 = ", (5 - 4)
123   print "(a8,i2)", "5 * 4 = ", (5 * 4)
124   print "(a8,i1)", "5 / 4 = ", (5 / 4)
125   ! Modulus
126   print "(a8,i1)", "5 % 4 = ", mod(5,4)
127   ! Exponentiation
128   print "(a7,i3)", "5**4 = ", (5**4)
129
130   ! You get 6 digits of precision by default
131   print "(f17.15)", float_num + float_num2
132
133   ! Doubles are accurate to 15 digits
134   print "(f18.16)", dbl_num + dbl_num2
135
136   Generate random values between 1 and 10
137   call random_number(rand)
138   print "(i2)", low + floor((high + 1 - low)*rand)
```

```
139
140   ! ----- Math Functions -----
141   print "(a10,i1)", "ABS(-1) = ", ABS(-1)
142   print "(a11,f3.1)", "SQRT(81) = ", SQRT(81.0)
143   print "(a9,f7.5)", "EXP(1) = ", EXP(1.0)
144   print "(a12,f7.5)", "LOG(2.71) = ", LOG(2.71)
145   print "(a12,i1)", "INT(2.71) = ", INT(2.71)
146   print "(a13,i1)", "NINT(2.71) = ", NINT(2.71)
147   print "(a14,i1)", "FLOOR(2.71) = ", FLOOR(2.71)
148   print "(a15,f3.1)", "MAX(2.7,3.4) = ", MAX(2.7,3.4)
149   print "(a15,f3.1)", "MIN(2.7,3.4) = ", MIN(2.7,3.4)
150   ! Trig functions use radians
151   print "(a14,f3.1)", "SIN(1.5708) = ", SIN(1.5708)
152   print "(a14,f3.1)", "COS(1.5708) = ", COS(1.5708)
153   print "(a14,f3.1)", "TAN(1.5708) = ", TAN(1.5708)
154   print "(a10,f3.1)", "ASIN(0) = ", ASIN(0.0)
155   print "(a10,f3.1)", "ACOS(0) = ", ACOS(0.0)
156   print "(a10,f3.1)", "ATAN(0) = ", ATAN(0.0)
157
158   ! ----- CONDITIONALS -----
159   ! Relational Operators : == /= > < >= <=
160   ! Logical Operators : .and. .or. .not.
161
162   ! If, else if, else
163   integer :: age = 16
164   if ((age >= 5) .and. (age <= 6)) then
165     print *, "Kindergarten"
166   else if ((age >= 7) .and. (age <= 13)) then
167     print *, "Middle School"
168   else if ((age >= 14) .and. (age <= 18)) then
169     print *, "High School"
170   else
171     print *, "Stay Home"
172   end if
173
174   print *, .true. .or. .false.
175   print *, .not. .true.
176   print *, 5 /= 9
177
178   ! Can be used with letters
179   print *, "a" < "b"
180
181   ! Select
182   select case (age)
183   case (5)
184     print *, "Kindergarten"
185   case (6:13)
186     print *, "Middle School"
187   case (14,15,16,17,18)
188     print *, "High School"
189   case default
190     print *, "Stay Home"
191   end select
192
193   ! ----- LOOPING -----
194   integer :: n = 0, m = 1
195   integer :: secret_num = 7
196
197   ! Start, Finish, Step
198   do n = 1, 10, 2
199     print "(i1)", n
200   end do
201
202   ! Exit & Cycle
203   ! Print only evens
204   do while (m < 20)
205     if (MOD(m,2) == 0) then
206       print "(i1)", m
207       m = m + 1
```

```
      ! Jumps back to beginning of loop
      cycle
    end if
    m = m + 1
    if (m >= 10) then
      ! Exits the loop all together
      exit
    end if
  end do

  ! Continue looping while a condition is true
  do while (n /= secret_num)
    print *, "What's your guess "
    read *, n
  end do
  print *, "You guessed it!"

  ! ----- ARRAYS -----
  ! Create ARRAY
  integer, dimension(1:5) :: a1, a2, a3
  real, dimension(1:50) :: aR1
  ! Create multidimensional array (Matrix)
  integer, dimension(5,5) :: a4
  integer :: n, m, x, y

  Define an array thats size is determined
  at run time
  integer, dimension(:), allocatable :: a5
  integer :: num_vals = 0

  integer, dimension(1:9) :: a6 = (/ 1,2,3,4,5,6,7,8,9 /)
  integer, dimension(1:3,1:3) :: a7

  ! Assign values (Starts at index 1)
  a1(1) = 5
  ! Retrieve value
  print "(i1)", a1(1)

  ! Assign values with a loop
  do n = 1,5
    a1(n) = n
  end do
  do n = 1,5
    print "(i1)", a1(n)
  end do

  ! Get a range
  print "(3i2)", a1(1:3)

  ! Get a range with an increment
  print "(2i2)", a1(1:3:2)

  ! Assign values to a multidimensional array
  do n = 1,5
    do m = 1, 5
      a4(n,m) = n
    end do
  end do
  do n = 1,5
    do m = 1, 5
      print "(i1,a1,i1,a3,i1)", n, " ", m, " : ", a4(n,m)
    end do
  end do

  ! Use an implied do loop to print each row
  ! on one line
  do n = 1,5
    print "(5i1)", ( a4(n,m), m = 1,5 )
  end do
```

```
277
278   ! Get size
279   print "(i2)", Size(a1)
280   print "(i2)", Size(a4)
281
282   ! Number of dimensions
283   print "(i2)", Rank(a4)
284
285   ! Elements in each dimension
286   print "(i2)", Shape(a4)
287
288   ! Define array size at run time
289   print *, "Size of array? "
290   read *, num_vals
291   allocate(a5(1:num_vals))
292   do n = 1,num_vals
293     a5(n) = n
294   end do
295   do n = 1,num_vals
296     print "(i1)", a5(n)
297   end do
298
299   ! Change all values in array
300   a2 = (/1,2,3,6,7/)
301
302   ! Implied do loop
303   print "(5i1)", ( a2(m), m = 1,5 )
304
305   ! Reshape the ARRAY from 1x9 t0 3x3
306   a7 = reshape(a6, (/ 3, 3 /))
307
308   ! Check if values are equal across
309   ! the 1 dimension
310   print "(l1)", all(a1==a2, 1)
311
312   ! Are any equal?
313   print "(l1)", any(a1==a2, 1)
314
315   ! How many are equal
316   print "(i1)", count(a1==a2, 1)
317
318   ! Get min and max value
319   print "(i1)", maxval(a1)
320   print "(i1)", minval(a1)
321
322   ! Get product and sum
323   print "(i3)", product(a1)
324   print "(i2)", sum(a1)
325
326   ! ----- FORMAT -----
327   ! The format statement has a numbered
328   ! label. You pass values to it that will
329   ! fit into the designated formatting
330   integer :: num
331   integer :: cups
332   real :: liters
333   real :: quarts
334
335   ! Print values 1 - 12 * 7
336   do num = 1,12
337     print 100, num, num * 7
338
339     ! I designates an integer along with
340     ! total space with values right justified
341     100 format(I2,' * 7 = ',I3)
342   end do
343
344   ! / Adds a newline
345   print "(/a18)", "Cups Liters Quarts"
```

```
  do cups = 1, 10
    liters = cups * .236
    quarts = cups * .208
    print 200 , cups,liters,quarts

    ! x defines spaces f is for floats
    200 format(' ',i3, 2x, f5.3, 2x, f5.3)
  end do

  ! ----- STRINGS -----
  ! Strings are character arrays
  character (len=30) :: str = "I'm a string"
  character (len=30) :: str2 = " that is longer"
  character (len=30) :: str3

  ! Join strings that have been trimmed of
  ! whitespace
  ! You can also trim right (adjustr) and
  ! left (adjustl)
  str3 = trim(str) // trim(str2)
  print *, str3

  ! Get a substring
  print *, str3(1:3)

  ! Find the index of a substring
  print "(a9,i1)", "Index at ", index(str, "string")

  ! Get size
  print *, len(str)

  ! Get number of items separated by a space
  print *, count_items(str)

  ! ----- STRUCTURES -----
  ! You can define custom types which contain
  ! multiple values of different types
  type Customer
    character (len = 40) :: name
    integer :: age
    real :: balance
  end type Customer

  type(Customer), dimension(5) :: customers

  integer :: n

  ! Create a customer
  type(Customer) :: cust1

  ! Assign values
  cust1%name = "Sally Smith"
  cust1%age = 34
  cust1%balance = 320.45

  ! Assign structure to array
  customers(1) = cust1

  ! Assign values independently
  customers(2)%name = "Tom May"
  customers(2)%age = 42
  customers(2)%balance = 229.78

  do n = 1, 2
    print *, customers(n)
  end do

  ! ----- FUNCTIONS -----
  ! Functions contain statements that return
```

```
  ! a single value

  integer :: ans, ans2
  real :: r_ans

  ans = get_sum(5,4)
  print "(a8,i1)", "5 + 4 = ", ans
  print "(a8,i1)", "5 + 4 = ", get_sum2(5,4)
  print "(a8,i1)", "5 + 4 = ", get_sum3(5)

  ! Use generic functions in a module that
  ! can work with ints and reals using
  ! the same function
  print "(a8,i2)", "5 * 4 = ", mult(5,4)
  r_ans = mult(5.3,4.4)
  print "(a12,f6.2)", "5.3 * 4.4 = ", r_ans

  ! Defines area for functions
  contains

    ! Return type, function, name, arguments
    integer function get_sum(n1, n2)
      implicit none
      integer :: n1, n2, sum

      ! The last value defined is returned
      sum = n1 + n2
    end function get_sum

    ! Define variable to be returned
    function get_sum2(n1, n2) result(sum)
      implicit none

      ! Don't allow variable values to change
      integer, intent(in) :: n1, n2
      integer :: sum
      sum = n1 + n2
    end function get_sum2

    ! Block functions from changing input
    ! variables with pure
    pure function get_sum3(n1, n2) result(sum)
      implicit none
      integer, intent(in) :: n1

      ! Arguments don't need to have a value passed
      integer, intent(in), optional :: n2
      integer :: sum

      if(present(n2)) then
        sum = n1 + n2
      else
        sum = n1 + 1
      end if
    end function get_sum3

  ! ----- RECURSIVE FUNCTIONS -----
  ! Recursive functions call themselves
  ! and must be labeled as such in Fortran
  integer :: ans
  ans = factorial(4)
  print "(a15,i3)", "Factorial(4) = ", ans

  ! 1st : result = 4 * factorial(3) = 4 * 6 = 24
  ! 2nd : result = 3 * factorial(2) = 3 * 2 = 6
  ! 3rd : result = 2 * factorial(1) = 2 * 1 = 2

contains

```

```
  recursive function factorial(n) result(o)
    integer :: n, o
    if (n == 1) then
      o = 1
    else
      o = n * factorial(n - 1)
    end if
  end function

  ! ----- SUBROUTINES -----
  ! Subroutines can return multiple values

integer :: i = 1, p1, p2
call plus_two(i, p1, p2)
print "(i1,/,i1,/,i1)", i, p1, p2

contains

  subroutine plus_two(n, plus1, plus2)
    integer, intent(in) :: n
    integer, intent(out) :: plus1, plus2 ! Output
    plus1 = n + 1
    plus2 = n + 2
  end subroutine plus_two

  ! ----- POINTERS -----
  ! Declare a pointer to an integer
  integer, pointer :: ptr1, ptr2

  ! Pointer to an array
  integer, pointer, dimension(:) :: a_ptr1

  ! Declare a target whose value changes
  ! as the pointers value changes
  integer, target :: target1

  ! Allocate space for a pointer
  allocate(ptr1)
  ptr1 = 5
  print "(a5,i1)", "ptr1 ", ptr1

  ! Associate pointer with target
  ptr2 => target1
  ptr2 = 1

  ptr2 = ptr2 + 2
  print "(a5,i1)", "ptr1 ", ptr1
  print "(a5,i1)", "tar1 ", target1

  ! Disassociate pointer and target
  nullify(ptr2)

  ! Deallocate storage for pointer
  deallocate(ptr1)

  ! ----- FILE I/O -----
  character (len=100) :: str = "I'm a string"
  character (len=100) :: str2

  ! If set to anything other then 0 an
  ! error occurred when opening a file
  integer :: err_status

  ! Used to catch error messages
  CHARACTER(256) :: err_iomsg

  ! Open / Create a FILE
  ! The unit number must be unique for
  ! each file
```

```
553   ! new (new file), old (exists),
554   ! scratch (file deleted after use)
555   open(10, file='data.dat', status='new', iostat = err_status, iomsg=err_iomsg)
556   if(err_status /= 0) then
557     write (*,*) 'Error ', trim(err_iomsg)
558
559     ! Stop execution
560     Stop
561   end if
562
563   ! Write string to file
564   write (10, '(A)') str
565
566   ! Close the file
567   close(10)
568
569   ! Open to read
570   open(11, file='data.dat', status='old')
571
572   ! Read from file
573   read (11, '(A)') str2
574   write (*, '(A)') trim(str2)
575
576   ! Either KEEP or DELETE file when closed
577   close(11, status="DELETE")
578
579   ! ----- ANOTHER MODULE EXAMPLE -----
580   ! Compile
581   ! gfortran -c shape.f90  fortrantut.f90
582   ! gfortran shape.o  fortrantut.o
583   ! ./a.out
584   call set_shape(10.5,20.5)
585   call get_area()
586
587   ! Compile
588   ! gfortran -c shape_mod.f90 triangle_mod.f90  fortrantut.f90
589   ! gfortran shape_mod.o triangle_mod.o  fortrantut.o
590   type(triangle_m) :: tri
591   tri%x = 10
592   tri%y = 20
593   print "(a3,f5.2)", "X: ", tri%x
594   print "(a3,f5.2)", "Y: ", tri%y
595   print "(a6,f6.2)", "Area: ", tri%get_area()
596
597 ! Define the end of the program
598 end program fortrantut
```