random() gives randomly generated numbers in [0, 1).
To find the probability of an event $X$ (say “heads”) when running a process $P$ (say tossing a coin):
Run the process many times, say $n_{\text{runs}} = \inf = 10^{\ast\ast}6$.
Count the number, $\text{num}_x$, of times event $X$ occurs.
Approximately, the probability of $X = \frac{\text{num}_x}{n_{\text{runs}}}$.
But, instead, you must write $X = \frac{\text{real}(\text{num}_x)}{n_{\text{runs}}}$ since Fortran divides reals, not integers.
Correct this program. Should get 0.5, not 0.

!real_prob.f95
program real_prob
integer :: numx=5, nruns=10; real :: prob
prob=numx/nruns
!don't divide integers, use real() to make numx a real.
print *, 'The probability = ', prob
endprogram

Monty Hall Problem
There are three closed doors: doors 1, 2, 3. A car is hidden behind a randomly chosen door, call it \( \text{cardoor} \in \{1, 2, 3\} \). The other two doors have goats.

You choose one of the doors, call it \( \text{firstchoice} \). Since they all have the same probability of having a car, let’s suppose \( \text{firstchoice}=1 \).

Monty Hall, the game show host, opens one of the other two doors (doors 2 or 3) to reveal a goat.

You now have two choices:
- \( \text{stay} \) -- stay with your first choice and open door 1 or
- \( \text{switch} \) -- switch to the other closed door.

You win if the door you open has the car.

**Classwork 28.1(7) montyhall.f95**

First write a function \( \text{cardoor()} \) which randomly puts a card behind one of the three doors: doors 1, 2, 3

Write a function \( \text{winningchoice()} \) which shows which choice won: \( \text{stay} \) if the car is behind the door 1, \( \text{switch} \) if it is behind the other closed door. Run this a million times to find which choice wins most often.
!c28_1_7montyhall.f95

integer function cardoor()
! cardoor() randomly picks the door where the car is placed
real:: r; r=rand()
!~implicit none
if (r<1./3.) then; cardoor=1
elseif (_______) then; _________
else; _________
endif ! each door should have probability 1/3
print*, cardoor ! delete this line
endfunction

program cardoor_test
integer:: cardoor, nruns=10, numdoor (3)=
real:: prob (3)
! numdoor(1) = number of times car is behind door 1
! numdoor(i) = number of times car is behind door i, ...
do n=1, nruns
   i=cardoor(); numdoor(i)=
! if cardoor()==2, increase the count numdoor(2) for door 2.
enddo
prob =
print*, ' door1  door2  door3'
print ' (3(f6.2,"\% "))', prob (1:3)
If \texttt{cardoor()} picks doors randomly, the doors should be about equally likely.

Delete the \texttt{cardoor\_test} program and paste in the next function and program.

\begin{verbatim}
character(6) function winningchoice()
integer :: cardoor, firstchoice = 1
if(cardoor() == firstchoice) then
    winningchoice = ____  ! 'stay' or 'switch' ?
else; winningchoice = ____  ! 'stay' or 'switch' ?
endif !return which is the winning choice, '...' need the quotes.
endfunction
\end{verbatim}

\begin{verbatim}
program monty\_hall
character(6):: winningchoice
integer:: nstay, nswitch, nruns=10**6
real:: probstay, probswitch
nstay = _______  ! nstay = # of times staying wins, start count
\end{verbatim}
nswitch = _______ !nswitch = # of times switching wins

do i=1,nruns
  if(winningchoice()=='stay') then
    nstay = ______________
  else; nswitch = ______________
  endif
enddo

!probstay = probability staying wins
probstay = _________________

!probswitch = probability switching wins
probswitch = _________________

print *,'Prob. staying wins:',probstay
print *,'Prob. switching wins:',probswitch

endprogram ! □
**Random integers**

\( r = \text{rand}() \) is a random real in \([0,1)\).

\( 4r \) is a random real in \([0,4)\).

\( \text{floor}(4r) \) rounds these down to random integers 0, 1, 2, 3.

\( \text{floor}((n+1)r) \) rounds gives random integers 0, 1, 2, 3,..., \( n \).

\( \text{m+floor}((n+1-m)r) \). It has random integers \( m, m+1,..., n \).

Let \( \text{randint}(m,n) \) be these random integers from \( m \) to \( n \).

```fortran
!randint.f95
integer function randint(m,n)
randint = m+floor((n+1-m)*rand())
endfunction

program randint_test
integer::randint
do i=1,20;print*,randint(4,7);enddo
endprogram
```

\( \text{rand}() \) doesn’t work with the online compiler. But when user input is involved, the time in milliseconds after the last second (\texttt{values}(8) of the \texttt{date_and_time} function) when a keyboard entry is made is reasonably random. If you
use the online compiler and your program uses user input, use this instead.

!randint_user.f95

define function randint(m,n)
define::n,m,values(8)
define character(10)::date,time,zone
call date_and_time(date,time,zone,values)
define randint = m+mod(values(8),n+1-m)
! values(8)=milliseconds after current second.
define endfunction

define program randint_test

define::randint
define character(1)::a
do
   Print*,"Press any key. Press 0 to quit."
   read*,a; if(a=="0") exit
   print *,randint(4,7)
enddo
Print*
define endprogram
Suppose you want to randomly pick integers from 1, 2, 3, 4 but instead of their being equally likely, you want their probabilities to be $\text{prob} = [\text{prob}(1), \text{prob}(2), \text{prob}(3), \text{prob}(4)] = [.2, .2, 0., .6]$, then use the following $\text{randint}_\text{prob}(\text{prob}, n)$ function.
integer function rand_prob(prob,n)
real:: prob(n), r, p
!
! picks numbers from 1, ..., n, i has probability prob(i)
!
r=rand(); p=0
do i=1,n
  if(p<r .and. r<p+prob(i) )then
    rand_prob=i; return
  endif
  p=p+prob(i)
enddo
endfunction

program test_rand_choose
real:: prob(4)
integer:: i, rand_prob
!
prob=(/.2, .2, 0., .6/) !
do i=1,20;
  print*, rand_prob(prob,4)
enddo
endprogram
Given a deck of \( n \) cards, \( \text{cards} = [1,2,3,...,n] \), you must shuffle the deck, i.e., randomly permute the cards. If \( n = 52 \), the shuffled deck might look like \( \text{cards} = [28,4,49,33,8,...,6] \).

Algorithm: for each \( i \) from 1 to \( n \), use \( \text{randint}(i,n) \) to randomly pick a \( j \) in \( \{i, i+1,...,n\} \). Then swap cards in positions \( i \) and \( j \).
integer function randint(m, n)
    randint = m + floor((n + 1 - m) * rand())
endfunction

subroutine swap(i, j)
    k = i; i = j; j = k
endsubroutine

subroutine shuffle(cards, n)
integer:: cards(n), randint
    ... replace with lines needed to shuffle a deck of n cards
endsubroutine

program randint_test
integer:: cards(52), randint
do i = 1, 52; cards(i) = i; enddo
Print*,'First 13 cards of 5 shuffles.'
do i = 1, 5
    call shuffle(cards, 52)
    print '(13(i3),a)', (cards(j), j = 1, 13)," ..."
endoalgo
endprogram
Write a program which plays rock-paper-scissors. For the benefit of online, we use `date_and_time` for random values. `youtmove` is the move the user picks, `mymove` is the move the computer randomly generates. Use `randint` to randomly pick 1, 2, or 3 where 1 = rock, 2 = paper, 3 = scissors. \( \text{winner}(i,j) = i \) if \( i \) wins, \( j \) if \( j \) wins, 0 on a draw.

```fortran
!h28_1_3rockpapercissors.f95
integer function randint(m,n)
integer::n,m,values(8)
character(10)::date,time,zone
call date_and_time(date,time,zone,values)
randint = m+mod(values(8),n+1-m)
endfunction

program rockpapercissors
integer::youmove,mymove,randint
```

integer:: win, winner(3,3)
integer:: rock=1, paper=2, scissors=3, draw=0
character(8):: move(3)
move(1)='rock'; move(2)='paper'; move(3)='scissors'
!
winner(i,j) = i if i wins, j if j wins, draw if neither wins
!
winner(rock,:) = [draw, paper, rock]
winner(paper,:) = [paper, draw, scissors]
winner(scissors,:) = [scissors, paper, draw]

print*, 'Enter 0 to quit.'
do
  print*, &
  "Enter 1 for rock, 2 for paper, 3 for scissors.>"
  read*, youmove; if(youmove==___) exit;
  if(youmove>___) cycle  !cycle if not 1, 2, 3
  print*, 'you: ', trim(move(youmove))
  mymove=__________  !use randint to randomly pick 1,2,3
  print*, ' me: ', trim(move(mymove))
  win=winner(mymove, youmove)
  if(win==0)then;print*, 'Draw'; endif
if(win==youmove) then; print*,'You win'; endif
if(win==mymove) then; __________ endif
print*
enddo
endprogram

Suppose data is coming in a stream, 4, 2, 3, 2, 6, ... . And, we wish to keep a running average. As each new data item comes in, we need a way to update the average without retaking the total every time. Suppose the first \( n \) items are \( a_1, a_2, a_3, ..., a_{n-1}, a_n \). Suppose \( A_{n-1}, A_n \) are the respective averages of the first \( n - 1 \) and first \( n \) items. Then

\[
\frac{(A_{n-1}(n - 1) + a_n)}{n}
\]

\[
= \left( \frac{a_1 + a_2 + a_3 + ... + a_{n-1}}{n - 1} \right) \cdot \frac{(n - 1) + a_n}{n}
\]

\[
= \left( a_1 + a_2 + a_3 + ... + a_{n-1} + a_n \right) / n = A_n
\]
A drunk starts at the origin \([0,0]\). Each step is randomly 1 unit to the right, 1 unit up, 1 unit to the left or 1 unit down. On average, how far from the origin will he after 100 steps?

```
integer function randint(m,n)
    randint = m+floor((n+1-m)*rand())
endfunction

subroutine onestep(s)
    integer::s(2),i,randint
    i= _______  !use randint to pick i randomly from 1,2,3,4
    if(i==1) then;  s=[1,0]  !right
    elseif(i==2)then;  s=[0,1]  !up
    elseif(i==3)then;  s= _______  !left
    else;  s= _______  !down
    endif
endsubroutine

subroutine randomwalk(n,w)  !w= final position
    integer::s(2),w(2)
```
!random walks start at the origin

\[ w = \text{_____} \] 
\[
\text{do } i=1,n; \\
\text{_________} \] !use onestep to generate a random step
\[
\text{w = } \text{_______} \] !update the position by adding step s.
\text{enddo}
\text{endsubroutine}

real function distance(w)
integer::w(2)
distance=sqrt(real(w(1))**2+real(w(2))**2)
endfunction

real function average_dist(n)
!average distance from origin after n random steps
real::distance
integer::w(2),inf=10**3
\[
\text{do } i=1,inf \text{do } i=1,inf \\
\text{call randomwalk(n,w)} \\
\text{_____________} \] !update the running average
\text{enddo}
\text{endfunction}
program average_walk_distance
real::average_dist

do n=10,100,10
print*,'Distance after',n,'steps:',average_dist(n)
enddo

endprogram ! Distance after 100 steps rounds to 9

CLASSWORK 28.1(7) montyhall.f95 28.2(4) shuffle.f95 28.3(5) randomwalk.f95
e-mail: dale@math.hawaii.edu subject line: 190 c28(16)

HOMEWORK: 28.1(6) rockpapercissors.s95