```
#######################
#Introduction to SPLUS#
#######################
#(note # is a "comment" symbol)

#If you want to find out about a particular command,
#type "help"; for example

help(sum)
help("sum")

#to find out about the "sum" function or

help("%*%")

#to find out about the strange looking operator %*%.

####################################################

#1.  Declarations
#The way to declare a scalar quantity x equal to numerical value 6.24, say,
is

x<-6.24

#The "<-" symbol is the "define equal to" symbol.
#We can do the same for character strings

x<-"This is x"

#You can type these commands in the Commands Window
#(go to the Window pulldown, select "Commands Window")
#
#Alternately, you can create a "Script" file (such as this one)
#and then execute it by highlighting a command, and then either
#(i) pressing the F10 function button on the keyboard
#(ii) clicking the triangle button (just above where it says "Intro.ssc -
program"
#     at the top of the script window, or on the Splus button bar )

#To see what the object x contains type

x

#or

print(x)

#2.  Data Objects
#There are four basic types of Splus object that we will use
#(i) single values/scalars as above
#(ii) vectors
#(iii) matrices
#(iv) data frames

#(ii) We create a vector like this, using the concatenation function "c"

xvec<-c(1,2,3,4,5,10)

#Alternately

xvec<-c(1:5,10)

#The colon ":" symbol in this expression 1:5 means
#take all the numbers from 1 up to 5.


#Another way to create a vector is to use "rep".
#For example, for a vector of zeros of length 20

xvec<-rep(0,20)
```

```
#A further way to create a structured vector is to use the sequence
function "seq".

xvec<-seq(from=1,to=20,by=1)

#Elements of a vector are accessed using square bracket notation

xvec[10]

#In fact, all non-scalar quantities are referenced via square brackets


#(iii)  Matrices: we create a matrix using the "matrix" function

xmat<-matrix(0,nrow=10,ncol=4)

#xmat is a 10x4 matrix of zeros

#The third row of the matrix xmat is accessed by

xmat[3,]

#The second column is

xmat[,2]

#The (i,j)th entry is
i<-1
j<-2
xmat[i,j]

#for integers i and j - for xmat, we must have i between 1 and 10, j
between 1 and 4.

xmat[7,3]

#To create a matrix from a vector

xvec<-c(1:40)
xmat<-matrix(xvec,nrow=10,ncol=4,byrow=T)

#which means read in row by row - T here means "TRUE" -
#and which gives a different result from

xmat<-matrix(xvec,nrow=10,ncol=4,byrow=F)

#and, of course,a different result from

xmat<-matrix(xvec,nrow=5,ncol=8,byrow=T)

#A final way to construct matrices is by "binding" vectors together
#either as rows (using rbind) or columns (using cbind)

xvec.1<-c(1:10)
xvec.2<-c(11:20)
xmat<-rbind(xvec.1,xvec.2)

xmat<-cbind(xvec.1,xvec.2)

#Note: There is a difference between a vector of length n
#and a (1xn) matrix; Splus always regards a matrix object
#as a 2 dimensional entity


#(iii) Data frames
#Data frames are essentially special, more sophisticated types of matrix

xvec.1<-c(1:10)
xvec.2<-c(11:20)
xdataframe<-data.frame(xvec.1,xvec.2)

#To access the entries in a data frame, we can either use the
```

```
#standard matrix indexing

xdataframe[1,2]
xdataframe[4,]
xdataframe[,2]

#or the "names" of the columns in the dataframe, and a dollar symbol "$"

names(xdataframe)

xdataframe$xvec.1

#3. Operators
#The numerical operations of addition, multiplication etc are implemented
in the
#obvious ways for scalars

x<-2.3
y<-3.8
z<-x+y #Add
z<-x*y #Multiply
z<-x-y #Subtract
z<-x/y #Divide
z<-x^2 #Power (square)
z<-exp(x) #Exponential
z<-log(x) #Natural log
z<-log10(x) #Log base 10
z<-x %% 2 #Modulo

#For matrices, the symbol % is used in a different way

xmat<-matrix(c(1,2,3,4),nrow=2,byrow=T)
ymat<-matrix(c(1,2,3,4),nrow=2,byrow=T)

zmat<-xmat %*% ymat #Matrix multiply

#If you just use the command

zmat<-xmat*ymat

#then you get component by component multiplication

#Other useful operations with vectors and matrices are

xvec<-c(1:10)
xmat<-matrix(c(1,2,3,4),nrow=2,byrow=T)

tot<-sum(xvec) #Sum of entries
tot<-sum(xmat)

m<-mean(xvec) #arithmetic of a vector

rowtotal.1<-sum(xmat[1,])
coltotal.2<-sum(xmat[,2])

t(xmat) #Transpose

solve(xmat) #Inverse of xmat

#4. Logical Operators

2 == 2 #Equal to should return T for - True
2 != 2 #Not Equal To should return F for - False
2 <  2 #Less Than - False
2 <= 2 #Less Than or Equal to - True

#Logical indexing of elements in a vector
x<-c(1:200)
x[x %% 10 == 0]<-1000
```

```
#5. Plotting
#The plot function is used to produce point or line plots
#The points function is used to add points

x<-seq(from=0,to=100,by=0.1)
y<-x^2
plot(x,y)
plot(x,y,type="l")
plot(x,cos(x),type="l")

x<-seq(from=0,to=1,by=0.001)
plot(x,x*(1-log(x)),type="l")
xp<-seq(from=0,to=1,by=0.05)
points(xp,xp*(1-log(xp))*cos(xp))

#6.  Loops
#Loop construction proceeds as for most computer code

x<-rep(0,100)
for(i in 1:100){
        x[i]<-i
}

count<-0
while(count < 100){
        count<-count+1
        x[count]<-10*count
}

count<-0
x<-rep(0,100)
while(count < 100){
        count<-count+1
        if(count > 50 & count <= 75) next #move to the end of the loop
        x[count]<-count
        if(count == 90) break #Terminate the loop
}


#This Worksheet has been completed
```