Visual Fractions

R.W. Oldford

Fall 2015

Drawing circles

Here, we are going to use some plotting primitives from the `grid` in `R`.

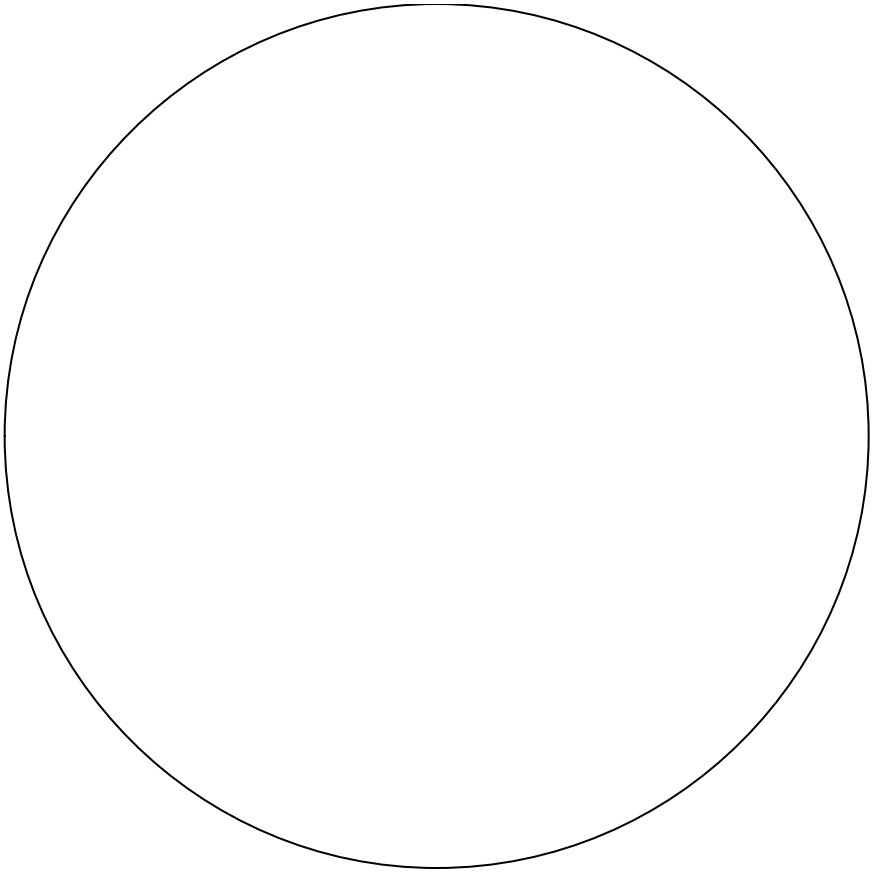
To begin, you will need to load `grid`. (It should already be installed as part of the `R` install.)

```
require("grid")
```

```
## Loading required package: grid
```

We're going to implement visual fractions using an array of circles. In grid this is simply done as follows:

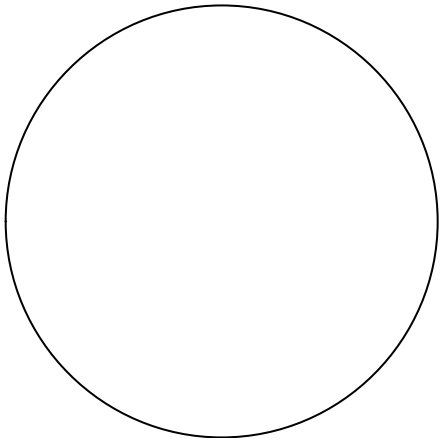
```
grid.circle()
```



Note that the circle occupies the whole of the plotting region. That's because in `grid`, the plotting region is a rectangular region which by default is a unit square from 0 to 1 on each side. This plotting region is called a `viewport`. Unless otherwise specified, a circle has centre `x=0.5`, `y=0.5` and radius `r=0.5`.

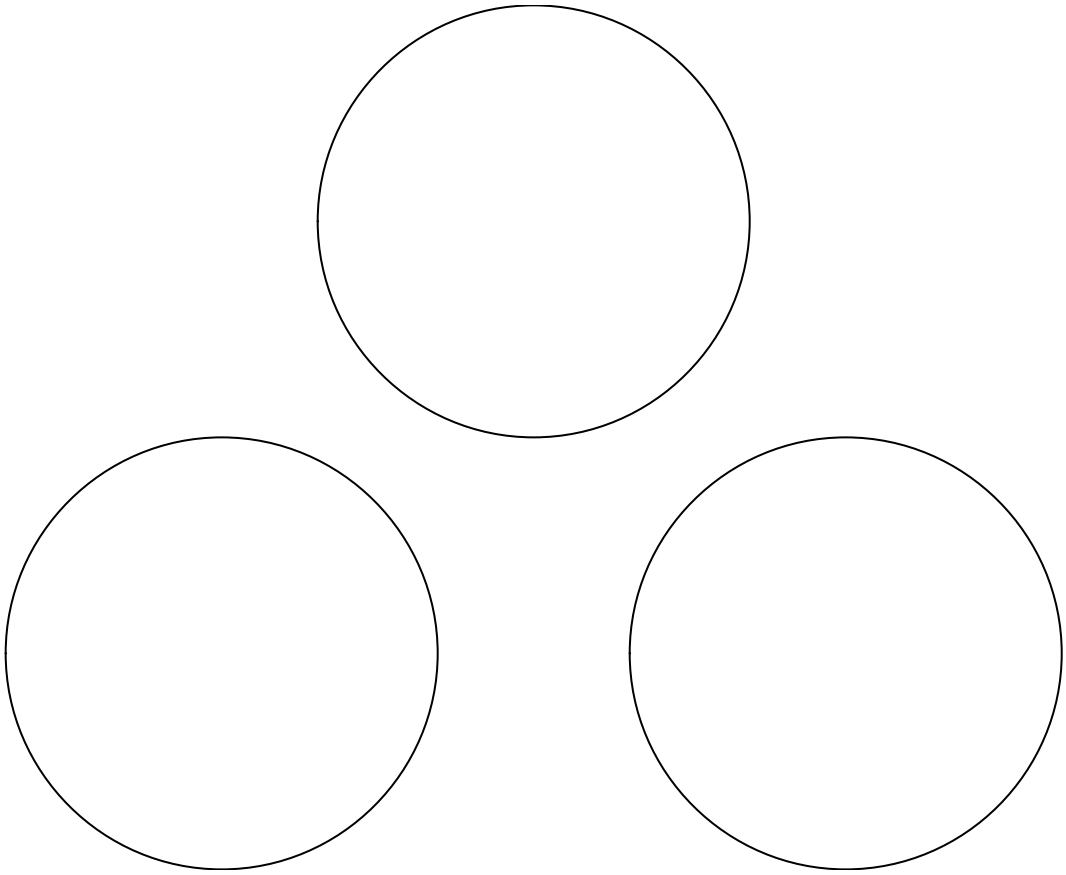
Here's another circle:

```
grid.circle(x=0.25,y=0.25,r=0.25)
```



Note that if we draw more than one, they all appear in the one plotting region:

```
grid.circle(x=0.25,y=0.25,r=0.25)
grid.circle(x=0.75,y=0.25,r=0.25)
grid.circle(x=0.50,y=0.75,r=0.25)
```



To get a new page on which to draw, we invoke

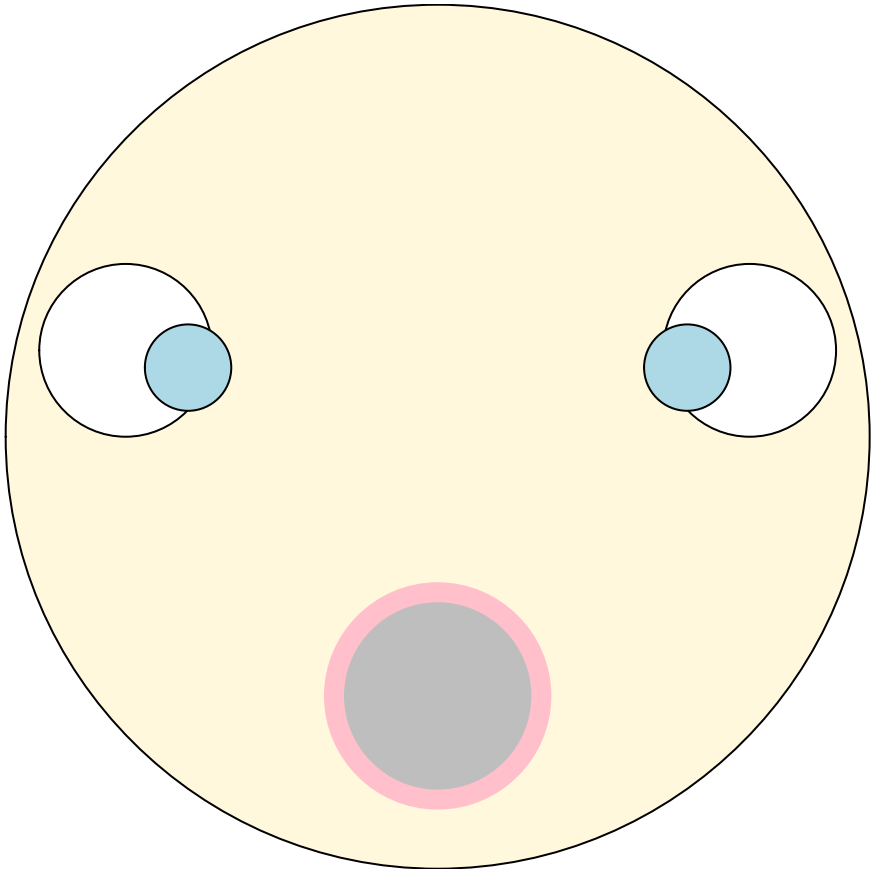
```
grid.newpage()
```

Graphical parameters

Each element drawn in grid has certain graphical parameters which can be set. The current default names and their values can be seen by executing `get.gpar()` or look at `help(gpar)`

We can, for example, add colours for the boundary (`col`), for the inside (`fill`), and other graphical parameters through the graphical parameter argument `gp` as in:

```
grid.newpage()
grid.circle(gp=gpar(fill="cornsilk"))
grid.circle(x=0.25,y=0.6,r=0.1,
            gp=gpar(fill="white"))
grid.circle(x=0.75,y=0.6,r=0.1,
            gp=gpar(fill="white"))
grid.circle(x=0.3,y=0.58,r=0.05,
            gp=gpar(fill="lightblue"))
grid.circle(x=0.7,y=0.58,r=0.05,
            gp=gpar(fill="lightblue"))
grid.circle(x=0.5,y=0.2,r=0.12,
            gp=gpar(fill="grey", col="pink", lwd=10))
```



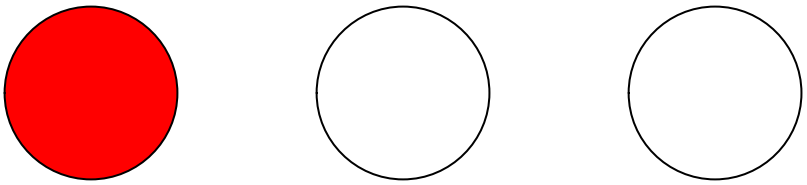
Note that when circles overlap, the last one plotted is on top. This is true, whatever graphical object (or **grob**) is being drawn, circles are just one example.

Visual fractions

A very simple way to produce visual fractions of the form $\frac{num}{den}$ is to plot an array of *den* non-overlapping circles all of one colour (e.g. `fill="white"`) and then overplot *num* of them using another colour (e.g. `fill="red"`). Or, avoid overplotting by simply choosing some subset of *num* of them to be plotted with the different colour to begin with.

For example, we could express the visual fraction $\frac{1}{3}$ by

```
grid.newpage()
grid.circle(x=1/4,y=0.5,r=0.1, gp=gpar(fill="red"))
grid.circle(x=1/2,y=0.5,r=0.1)
grid.circle(x=3/4,y=0.5,r=0.1)
```



Of course, if *den* was much larger we would need to arrange the circles in a matrix array.
To help simplify that job, it is easy to write a function that will return the appropriate (x, y) coordinates:

```
xy2grid <- function(x, y) {
  n <- length(x)
  m <- length(y)
  # Return the coordinates of the m x n grid havng
  # locations (x,y) for all x and y
  cbind(rep(x, times=m), rep(y, each=n))
}
#
# For example,
#
xy2grid(1:4, 10:12)
```

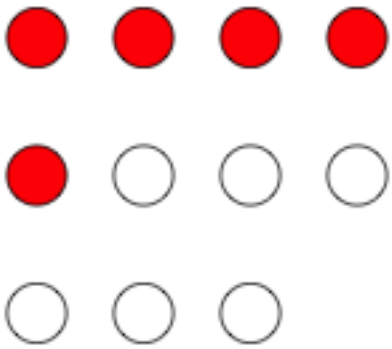
```
##       [,1] [,2]
##  [1,]    1   10
##  [2,]    2   10
##  [3,]    3   10
##  [4,]    4   10
##  [5,]    1   11
##  [6,]    2   11
##  [7,]    3   11
##  [8,]    4   11
##  [9,]    1   12
## [10,]    2   12
## [11,]    3   12
## [12,]    4   12
```

You now have all of the tools to write a function `visualFraction` which will draw a display of a visual fraction:

```
## The function will look like:
##
visualFraction <- function(num, # the numerator
                           den, # the denominator
                           numCol="red",
                           # numerator colour
                           denCol="white",
                           # denominator colour
                           random=FALSE,
                           # a logical indicating
                           # whether the numerator values
                           # are to appear at random
                           # locations (if TRUE) or not.
                           ncols = NULL
                           # number of columns to be
                           # used in the array
) {
  # begin with some error checking
  #
  #  Check the logical
  if (!is.logical(random))
    stop(paste("random must be TRUE or FALSE, not:",
               random))
  #
  #  Check the numerator
  if (!is.numeric(num))
    stop(paste("num must be a number, not", num))
  if (length(num) != 1)
    stop(paste("num must be a single number, not of length",
               length(num)))
  if (floor(num) != num | num < 0 )
    stop(paste("num must be a non-negative integer, not",
               num))
  #
  #  Check the denominator
  if (!is.numeric(den))
    stop(paste("den must be a number, not", den))
  if (length(den) != 1)
    stop(paste("den must be a single number, not of length",
               length(den)))
  if (floor(den) != den | den < 0 )
    stop(paste("den must be a non-negative integer, not",
               den))
  #
  #  Check both
  if (num > den)
    stop(paste("num =", num, "> den =", den))
  #
  #  Check ncols
  #
  #  Default is NULL, so if user doesn't supply one let's
  #  try to make it close to square (default more cols than rows)
  if (is.null(ncols)) ncols <- ceiling(sqrt(den))
```

```
  #  Now check any user supplied value for ncols
  if (!is.numeric(ncols))
    stop(paste("ncols must be a number, not", ncols))
  if (length(ncols) != 1)
    stop(paste("ncols must be a single number, not of length",
               length(ncols)))
  if (floor(ncols) != ncols | ncols < 0 )
    stop(paste("ncols must be a non-negative integer, not",
               ncols))
  if (ncols > den )
    stop(paste("ncols =", ncols,"> den =", den))

  ## If we have ncols columns, we will need
  ## nrows rows where
  nrows <- ceiling(den/ncols)

  ## We'll also need a radius
  ## This is size provides spacing for most
  radius <- 1/(2*(max(nrows,ncols)+5))

  ##
  ## Now it's your turn
  ## The display should be an nrows x ncols array of den circles
  ##
  ## If random=FALSE, the first num circles (from the top left of the
  ## array and proceeding left to right, then top to bottom)
  ## should be coloured numCol, the remainder coloured denCol.
  ##
  ## If random=TRUE, num circles selected at random in the array
  ## should be coloured numCol, the remainder denCol.
  ##
  ## That is, if we index the array 1 to den from top left by row to bottom
  ## right, the indices we would need to colour numCol would be
  if (random) {indices <- sample(1:den, num)} else {indices <- 1:num}
  ##
  ## INSERT YOUR CODE BELOW:

}
```

For example, the output of `visualFraction(5,11)` should be



In debugging your code, especially in `RStudio`, you might try `debug(visualFraction)` to step through the code with repeated returns. At any point you can evaluate the various symbols to aid in the debugging. You can turn it off with `undebug(visualFraction)`.