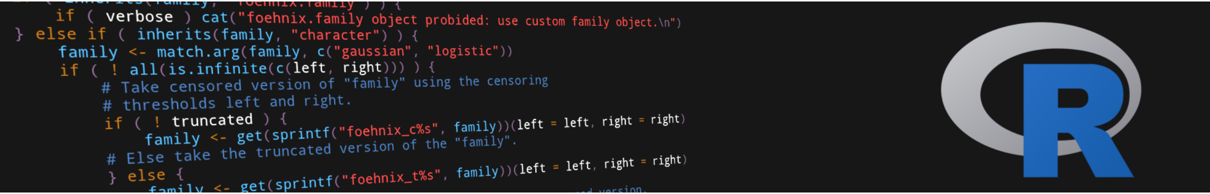


```
    if ( verbose ) cat("foehnix.family object probided: use custom family object.\n")
} else if ( inherits(family, "character") ) {
    family <- match.arg(family, c("gaussian", "logistic"))
    if ( ! all(is.infinite(c(left, right))) ) {
        # Take censored version of "family" using the censoring
        # thresholds left and right.
        if ( ! truncated ) {
            family <- get(sprintf("foehnix_c%s", family))(left = left, right = right)
        # Else take the truncated version of the "family".
        } else {
            family <- get(sprintf("foehnix_t%s", family))(left = left, right = right)
```

Flow control

if

```
if (cond) {
    expr
}
```

```
if (cond) {
    expr1
} else {
    expr2
}
```

```
if (cond1) {
    expr1
} else if (cond2) {
    expr2
} else {
    expr3
}
```

- The `cond` should evaluate to a single `TRUE` or `FALSE`.
- There is no `elseif`, but `else if` has to be used.
- `if` returns a value and thus can be used inline.
- Use `ifelse()` for vectorized conditions.
- Curly brackets `{}` can be omitted for single statments.

if

```
R> x <- 2
R> if (x %% 2 == 0) {
+       cat("The number", x, "is even.")
+  } else {
+       cat("The number", x, "is odd.")
+  }
```

```
The number 2 is even.
```

```
R> cat("The number", x, "is", if (x %% 2 == 0) "even." else "odd.")
```

```
The number 2 is even.
```

```
R> x <- 1:10
R> ifelse(x %% 2 == 0, "even", "odd")
```

```
 [1] "odd"  "even" "odd"  "even" "odd"  "even" "odd"  "even"
 [9] "odd"  "even"
```

Conditions

- **Relational operators**: `<`, `>`, `<=`, `>=`, `==`, `!=`
- **Logical operators**: `!`, `&`, `|`, `xor()`, `&&`, `||`
- **Value matching**: `%in%`, `match()`.
- `any()` **and** `all()`.

Relational operators

```
R> x <- 1:3
R> y <- c(1:2, 4)
R> x == y

[1]  TRUE  TRUE FALSE

R> x >= 2

[1] FALSE  TRUE  TRUE
```

Logical operators

```
R> (x == y) & (x >= 2)
[1] FALSE  TRUE FALSE
R> xor(!(x == y), !(x >= 2))
[1]  TRUE FALSE  TRUE
R> (x == y) && (x >= 2)
[1] FALSE
R> is.character(x) && (x == "hallo") && stop("This is an error!")
[1] FALSE
```

- `&&` and `||` for `if` and `while` conditions.
- `&` and `|` for subsetting with logical vectors or `ifelse`.

Value matching

```
R> "A" %in% LETTERS
[1] TRUE
R> 1 %in% 1:10
[1] TRUE
R> c("A", "Z", "AA") %in% LETTERS
[1]  TRUE  TRUE FALSE
R> match(c("A" , "Z", "AA"), LETTERS)
[1]  1 26 NA
R> match(c("A" , "Z", "AA"), LETTERS, nomatch = 0) > 0
[1]  TRUE  TRUE FALSE
```

- `%in%` with a single-valued vector on the left hand side for `if` and `while` conditions.
- `%in%` with vectors on the left hand side for subsetting with logical vectors or `ifelse`.
- `match` for subsetting with integer vectors.

any and all

```
R> any(c("A", "Z", "AA") %in% LETTERS)
[1] TRUE
R> any(c("A", "B", "C") %in% LETTERS)
[1] TRUE
R> all(c("A", "Z", "AA") %in% LETTERS)
[1] FALSE
R> all(c("A", "B", "C") %in% LETTERS)
[1] TRUE
```

for loops

```
for (var in seq) {
    expr
}
```

- ‘seq‘ is an atomic vector or a list.
- Within the loop ‘var‘ will take the values of ‘seq‘ iteratively.
- Use ‘next‘ to exit the current iteration.
- Use ‘break‘ to abort the entire for loop.

for loops

Example: Fill a numeric vector in a for loop:

```
R> fibonacci <- numeric(20)         ## initialize empty vector
R> fibonacci[1:2] <- c(0, 1)
R> head(fibonacci)
```

```
[1] 0 1 0 0 0 0
```

```
R> for (i in 3:20) {
+      fibonacci[i] <- fibonacci[i - 1] + fibonacci[i - 2]
+  }
R> head(fibonacci)
```

```
[1] 0 1 1 2 3 5
```

```
R> tail(fibonacci)
```

```
[1]  377  610  987 1597 2584 4181
```

while loops

```
while (cond) {
    expr
}
```

- ‘expr‘ is repeated as long as ‘cond‘ is ‘TRUE‘.

while loops

Example: Approximate golden ratio:

```
R> eps <- Inf
R> golden_ratio <- Inf
R> fibonacci_m1 <- 0
R> fibonacci <- 1
R> k <- 1
```

```
R> print((1 + sqrt(5)) / 2, digits = 20)
```

```
[1] 1.6180339887498949025
```

while loops

Example: Approximate golden ratio:

```
R> while (eps > 0.01) {
+      k <- k + 1
+      # --- update fibonacci numbers ---
+      fibonacci_m2 <- fibonacci_m1
+      fibonacci_m1 <- fibonacci
+      fibonacci <- fibonacci_m1 + fibonacci_m2
+      # --- update golden ratio ---
+      golden_ratio_m1 <- golden_ratio
+      golden_ratio <- fibonacci / fibonacci_m1
+      eps <- abs(golden_ratio - golden_ratio_m1)
+      cat(sprintf("%d: fib % 8d gr %0.5f eps %0.5f\n", k, fibonacci, golden_ratio, eps))
+  }
2: fib        1 gr 1.00000 eps Inf
3: fib        2 gr 2.00000 eps 1.00000
4: fib        3 gr 1.50000 eps 0.50000
5: fib        5 gr 1.66667 eps 0.16667
6: fib        8 gr 1.60000 eps 0.06667
7: fib       13 gr 1.62500 eps 0.02500
8: fib       21 gr 1.61538 eps 0.00962
```

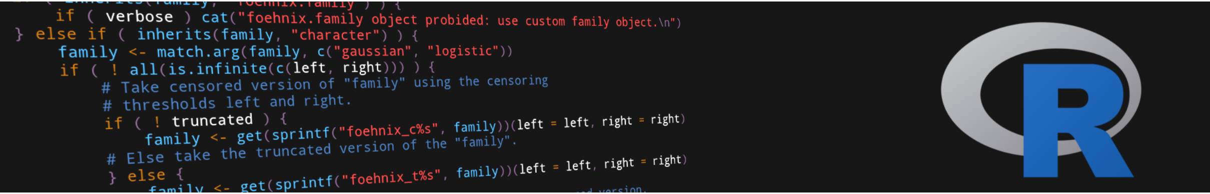
Style guide for code blocks

Curly brackets `{}` define the most important hierachy of R code. To make this hierachy easy to read, follow these guidelines:

- After starting the code block with `if`, `for`, `while` or `function`, the opening brace `{` should be the last character on the line.
- The contents should be intended by four spaces.
- The closing brace `}` should be in a newline, un-intended and the first character of that line.

Spacing:

- Place a space before and after `()` when used with `if`, `for` and `while`.
- Place a space before and after operators such as `<`, `>`, `<=`, `>=`, `==`, `!=`, `&`, `|`, `&&`, `||`, and `%in%`.



```
    if ( verbose ) cat("foehnix.family object probided: use custom family object.\n")
} else if ( inherits(family, "character") ) {
    family <- match.arg(family, c("gaussian", "logistic"))
    if ( ! all(is.infinite(c(left, right))) ) {
        # Take censored version of "family" using the censoring
        # thresholds left and right.
        if ( ! truncated ) {
            family <- get(sprintf("foehnix_c%s", family))(left = left, right = right)
        # Else take the truncated version of the "family".
        } else {
            family <- get(sprintf("foehnix_t%s", family))(left = left, right = right)
```

Functions

Functions: Basics

```
01  roll <- function(pips = 1:6) {
02      dice <- sample(pips, size = 2, replace = TRUE)
03      return(sum(dice))
04  }
```

- **Name**: `roll`. For calling the function.
- **Arguments**: `pips`. For providing values to the function.
- **Default values**: `= 1:6`. The value of the argument, if not specified differently.
- **Body**: Line 02 and 03. List of commands inside the function.
- **Last line of body**: The value of the last line of code is returned by the function. Use `return()` for explicit returning.

Functions: Basics

```
R> roll()
```

```
[1] 9
```

```
R> x <- numeric(10000)
R> for (i in seq_along(x)) {
+     x[i] <- roll()
+  }
R> head(x, 20)
```

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

```
R> round(prop.table(table(x)) * 36)
```

```
x
 2  3  4  5  6  7  8  9 10 11 12
 1  2  3  4  5  6  5  4  3  2  1
```

Example: Seven eleven

We want to write a function `seven_eleven` that implements the rules of *Seven Eleven* and executes one round of the game:

- Roll two dice a first time:
  - You win given 7 or 11 points.
  - You lose given 2, 3 or 12 points.
  - If you roll something else the points are called **point**.
- Keep rolling the dice until
  - you roll again the **point**, then you win,
  - or a 7, then you lose.

The function needs no input arguments and should return a numeric `1` if you win or a `0` if you loose.

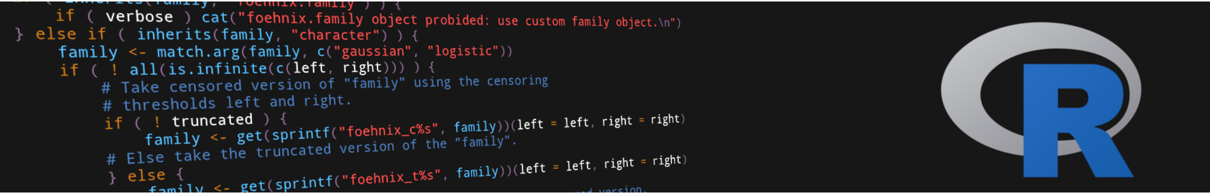
Collect the functions `roll()` and `seven_eleven()` in an R script called `04_<familyname>.R`, so that you can `source()` it.

Implementation

```r
seven_eleven <- function() {
    point <- roll()
    if (point %in% c(7, 11)) {
        rval <- 1
    } else if (point %in% c(2, 3, 12)) {
        rval <- 0
    } else {
        rval <- -1
        while (rval == -1) {
            points <- roll()
            if (points == point) {
                rval <- 1
            } else if (points == 7) {
                rval <- 0
            }
        }
    }
    return(rval)
}
```

Enter the casino

```
R> seven_eleven()

[1] 1

R> system.time( x <- replicate(10000, seven_eleven()) )

   user  system elapsed
  0.165   0.000   0.165

R> head(x, 20)

 [1] 1 0 1 0 0 1 1 1 0 1 0 0 1 0 0 0 0 1 0 0

R> prop.table(table(x))

x
     0      1
0.5087 0.4913

R> wiki_value <- 244/495
R> wiki_value

[1] 0.4929
```



```
if ( inherits(family, "foehnix.family") ) {
    if ( verbose ) cat("foehnix.family object probided: use custom family object.\n")
} else if ( inherits(family, "character") ) {
    family <- match.arg(family, c("gaussian", "logistic"))
    if ( ! all(is.infinite(c(left, right))) ) {
        # Take censored version of "family" using the censoring
        # thresholds left and right.
        if ( ! truncated ) {
            family <- get(sprintf("foehnix_c%s", family))(left = left, right = right)
        # Else take the truncated version of the "family".
        } else {
            family <- get(sprintf("foehnix_t%s", family))(left = left, right = right)
```

For loop replacements