Ex1

```
fun1<-function(x){
n<-length(x)
x^(1:n)
}
```

```
fun2<-function(x){
n<-length(x)
x^(1:n)/(1:n)
}
```

Representação gráfica:

```
plot(-4:1, fun1(-4:1), type="l")
lines(-4:1,fun2(-4:1),col="red")
```

Ex2

(a) Para definir a função pretendida sem inclusão de mensagem de erro fazemos:

```
f1<-function(x,y){
n=length(x)
y[2:n]-x[1:(n-1)]
}
```

Para incluir uma mensagem de erro usamos a instrução

```
if (cond) {bloco de instruções} else {bloco de instruções}
```

em combinação com a função stop() que vai parar a execução e apresentar a mensagem indicada entre os parentêsis. Note que a condição no "if else" deve ser um vetor lógico de comprimento 1.

Assim no exemplo, fazemos:

```
f1<-function(x,y){
if (length(x)!=length(y)) {stop("comprimentos diferentes")} else{
n=length(x)
y[2:n]-x[1:(n-1)]
}
}
```

b)

```
f2<-function(x,y){
if (length(x)!=length(y)) {stop("comprimentos diferentes")} else{
n=length(x)
exp(y[1:(n-1)])*x[2:n]
}
}
```

Ex 3

```
f3<-function(x){
if(length(x)<3) {stop("vetor de comprimento <3")} else{
n=length(x)
x[1:(n-2)]+2*x[2:(n-1)]-x[3:n]
}
}
```

```
f4<-function(x){
if(length(x)<2) stop("vetor de comprimento <2") else{
n=length(x)
sum(exp(x[2:n])/(x[1:(n-1)]+10))
}
}
```

Ex 4

Para testar se um inteiro k é um divisor de n podemos fazer o teste

```
n%%k==0
```

Tome por exemplo n=12, k=3 e depois k=5.

```
12%%3==0
12%%5==0
```

Podemos fazer este teste para todos os inteiros de 1 até 12 fazendo

```
k<-1:12
12%%k==0
```

e vamos selecionar os divisores de 12 fazendo

```
k[12%%k==0]
```

Assim mais geralmente definimos:

```
div<-function(n){
k<-1:n
k[n%%k==0]
}
```

Fazendo, por exemplo, div(108) obtém-se os divisores de 108.

Para incluir uma mensagem de erro observamos que um número (real) n é um número natural se for positivo (digamos aqui >0) e se o resto da sua divisão inteira por 1 é igual a 0, ou seja se a condição n%%1==0 for verdadeira.

Assim fazemos

```
div<-function(n){
if(n<=0|n%%1!=0) stop("não é natural") else{
k<-1:n
k[n%%k==0]
}
}
```

Um primo é um número natural que tem exatamente 2 divisores, ou seja cujo vetor dos divisores tem comprimento 2. Assim definimos:

```
primo<-function(n){
length(div(n))==2
}
```

Teste se 347, 403 são primos.

Ex5

Informação sobre a função curve.

A função curve permite obter o gráfico de funções. A síntaxa básica é a seguinte:

```
curve(expr, from= , to= )
```

Aqui expr pode ser o nome de uma função já definida ou uma expressão que deve ser dada em função da letra x. Os campos from= e to= permitem indicar o intervalo pretendido.

Assim para obter o gráfico de f(x)=x^2 no intervalo [-2,2] podemos fazer:

```
curve(x^2,from=-2,to=2)
```

ou

```
f<-function(x){x^2}
curve(f,from=-2, to=2)
```

Nesta última forma, a letra usada para a variável de f poderia ser qualquer.

A função curve tem um campo n por defeito igual a 101. Para perceber o funcionamento da função curve
observe os gráficos dados pelos seguintes comandos (sendo f<-function(x){x^2})

```
curve(f,from=-2,to=2, n=3)
curve(f,from=-2,to=2, n=10)
curve(f,from=-2,to=2, n=50)
```

Percebe-se que o gráfico é obtido considerando n pontos x_i equidistantes no intervalo [-2,2] e ligando os pontos (x_i,f(x_i)) por segmentos de reta. Ou seja, o gráfico dado por curve(f,from=-2,to=2, n=3) corresponde a fazer

```
x<-seq(-2,2,by=2)
```

```
plot(x,f(x),type="l")
```

A função curve é de alto nível (produz cada vez um novo gráfico). No entanto com a opção add=T torna-se numa função de baixo nível. Execute os seguintes comandos:

```
curve(x^2,from=-2,to=2)
curve(4-x^2,add=T)
```

No exercício o gráfico pretendido é obtido por:

```
curve(x^2*cos(x),from=-1,to=2)
```

ou, alternativamente,

```
g<-function(u){u^2*cos(u)}
curve(g,from=-1,to=2)
```

Exploraremos mais tarde a adição de legendas e uso da função locator.

Ex6

Em primeira abordagem, queríamos definir a função f do modo seguinte

```
f<-function(x){
if (x<0) {-x} else{ x^2}
}
```

No entanto ao avaliar esta função no vetor v<-c(-2,1) podemos observar que o resultado não é o pretendido

```
> f(v)
[1]  2 -1
Warning message:
In if (x < 0) { :
  the condition has length > 1 and only the first element will be used
```

A razão é que, como uma condição válida deve ser um vetor lógico de comprimento 1, apenas a primeira componente do vetor v<0 é considerada. Neste caso é TRUE pelo que o ramos "-x" vai ser aplicado a todas as componentes do vetor v. Da mesma forma (e em coerência com o funcionamento da função curve discutido no Ex 5) podemos constatar que curve(f, from=-2, to=2) não dá o resultado pretendido.

A função f assim definida não é vetorizada (não se aplica corretamente em todas as componentes do vetor). De modo a definir uma função vetorizada podemos usar o comando ifelse.

ifelse(vetor lógico, instrução para componentes TRUE, instrução para componentes FALSE)

Assim definimos f por

```
f<-function(x){ifelse(x<0,-x,x^2)}
```

e podemos verificar que f(v) tal como curve(f, from=-2, to=2) dão o resultado pretendido.

Finalmente adiciona-se um segmento de reta que junta os pontos (-2,2) e (2,4) por:

```
lines(c(-2,2),c(2,4))
```

Ex7

```
stat<-function(x){
u<-c(mean(x),sd(x))
names(u)<-c("media","desvio-padrao")
u
}
```

Aplica-se às 20 primeiras entradas do vetor rivers por:

```
stat(rivers[1:20])
```

Ex8

Sendo x e y dois vetores de mesmo comprimento, podemos calcular o declive b da reta dos mínimos quadrados por

```
sum((x-mean(x))*(y-mean(y)))/sum((x-mean(x))^2)
```

Podemos depois calcular a ordenada à origem a por

```
mean(y)-b*mean(x)
```

Sendo assim a função (digamos "retaMQ") pretendida na alínea (a) pode ser definida do modo seguinte (com adição de nomes para a apresentação dos resultados):

```
retaMQ<-function(x,y){
b<-sum((x-mean(x))*(y-mean(y)))/sum((x-mean(x))^2)
a<-mean(y)-b*mean(x)
u<-c(a,b)
names(u)<-c("ordenada à origem","declive")
u
}
```

(b) Introduzimos os dados por

```
femur<-c(38,56,59,64,74)
umero<-c(41,63,70,72,84)
```

Os parâmetros da reta dos mínimos quadrados associada são dados por

```
> retaMQ(femur,umero)
```

```
ordenada à origem          declive
        -3.659587          1.196900
```

Isto significa que o modelo associado que dá o comprimento do úmero (y) em função do comprimento do fémur (x) é dado por:

y=-3.659587+1.196900*x

Para representar gráficamente os pontos e adicionar a reta, fazemos:

```
> plot(femur,umero)
> abline(retaMQ(femur,umero))
```

Nota: abline(a,b) permite adicionar a um gráfico já existente a reta de equação y=a+bx.