FEUP	FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO Mestrado em Engenharia Informática e Computação Fundamentos da Programação	Auto Teste AT04 FACULTATIVO Duração: 60 min.

NOTAS IMPORTANTES:

1 - Deve respeitar rigorosamente os nomes dos procedimentos que são indicados bem como os formatos de saída dos resultados.

2 - Não use nunca carateres acentuados, nem nos nomes dos procedimentos nem dos parâmetros.

3 - Utilize comentários só "with Semicolons" e nunca "with a Box".

4 - O código desenvolvido durante a realização da prova, contido num único ficheiro com a extensão ".scm", deve ser submetido no Moodle usando o "link" correspondente à prova realizada. A não observação desta regra levará a que o código submetido não possa ser avaliado.

Exercícios sobre Recursividade

Exercício 1

O procedimento **imprime-na-base** tem 2 parâmetros, **num** e **base**, para os quais espera dois argumentos inteiros positivos, ambos na base 10. Este procedimento imprime **num** na base especificada por **base**.

```
(define imprime-na-base
  (lambda (num base)
    (cond ((< num base) (display num))
          (else
            (imprime-na-base (quotient num base) base)
            (display (remainder num base))))))
```

```
> (imprime-na-base 76 2)
1001100
> (imprime-na-base 76 3)
2211
> (imprime-na-base 76 10)
76
```

Observe agora o comportamento do procedimento **imprime-na-base**, quando o argumento correspondente à base é superior a 10.

```
> (imprime-na-base 76 11)
610                          <---- deveria ser 6a
> (imprime-na-base 76 16)
412                          <---- deveria ser 4c
```

Comece por tentar perceber os resultados produzidos com os argumentos 76 e 11, e 76 e 16.

O procedimento designado por **imprime-ate-base-20** deve responder corretamente até à base 20, ou seja, para além dos dígitos decimais (0 a 9) ainda utiliza as letras de a, b, c, d, e, f, g, h, i, e j.

```
> (imprime-ate-base-20 76 11)
6a                           <---- resposta correta...
> (imprime-ate-base-20 76 16)
4c                           <---- resposta correta...
> (imprime-ate-base-20 76 20)
3g                           <---- resposta correta...
> (imprime-ate-base-20 79 20)
3j                           <---- resposta correta...
> (imprime-ate-base-20 80 20)
40                           <---- resposta correta...
>
```

Alguém concluiu que o procedimento **imprime-ate-base-20** poderia resultar de pequenas alterações ao procedimento **imprime-na-base**. Observe que, em **imprime-na-base**, as duas situações em que **display** é chamado para visualizar um número, este número é OBRIGATORIAMENTE menor que a base. Então vamos começar por desenvolver o procedimento **decimal->outra** que recebe um número inferior à base (seja ela qual for até 20) e devolve:

Para números menores que 10,

```
> (decimal->outra 3)
3             ; devolve o próprio número.
```

Para números maiores que ou iguais a 10, e menores que 20,

```
> (decimal->outra 10)
"a"           ; devolve o dígito correspondente (sob a forma de uma cadeia com um caráter)
> (decimal->outra 19)
"j"
```

Para números maiores que ou iguais a 20,

```
> (decimal->outra 20)
"?"           ; devolve "?"
> (decimal->outra 30)
"?"
>
```

1.1 - Comece por escrever em Scheme o procedimento **decimal->outra**.
1.2 - Com base no procedimento **decimal->outra**, escreva em Scheme o procedimento **imprime-ate-base-20**.

Depósitos a prazo

Exercício 2
As taxas de juro pagas por um banco, nos depósitos a prazo, são variáveis ao longo do período do depósito. O procedimento **interest-rate** tem um parâmetro **n** e devolve o valor da taxa de juro paga pelo banco no **n**-ésimo ano do período, de acordo com os valores da tabela.

ano (n)	taxa de juro
1	2.0%
2	2.4%
3	3.2%
4 e seguintes	3.8%

```
> (interest-rate 1)
0.02
> (interest-rate 3)
0.032
> (interest-rate 4)
0.038
> (interest-rate 7)
0.038
```

2.1. Completar o procedimento **interest-rate**, admitindo que o parâmetro **n** tem sempre um valor válido (>=1):

```
(define interest-rate
 (lambda (n)
       ...
```

Exercício 3
O procedimento **accumulate** tem dois parâmetros, **amount** e **n-years**, representando o montante inicial depositado numa conta bancária e o número de anos em que esse montante esteve depositado, e devolve o montante total disponível após esse período, considerando que os juros vencidos em cada ano (calculados às taxas indicadas no exercício 2) são acumulados ao montante do depósito no final desse ano e, por isso, o montante total depositado vai crescendo de ano para ano. Caso algum dos parâmetros seja negativo ou nulo, o procedimento deve visualizar a mensagem **"invalid parameter(s)"**, não interessando o que devolve, neste caso. Os resultados devem ser apresentados arredondados às centésimas; o arredondamento só deve ser feito no final de todos os outros cálculos.

```
> (accumulate 100 0)   ; NOTA: accumulate escreve-se com 2 c's
invalid parameter(s)   ; NOTA: esta mensagem não deve ser seguida de (newline)
> (accumulate 100 1)
102.0
> (accumulate 50 2)
52.22
> (accumulate 2500 20)
5080.13
> (accumulate -500 6)
invalid parameter(s)
```

O procedimento **accumulate** vai necessitar de um procedimento auxiliar, recursivo, **accumulate-aux**, que tem um parâmetro adicional, **year**, que indica o ano relativamente ao qual se devem fazer os cálculos, em cada iteração.
Completar os procedimentos **accumulate** e **accumulate-aux**:

```
(define accumulate
 (lambda (amount n-years)
   (if ...   ; testa se os parâmetros são válidos
       ...
       (accumulate-aux amount n-years 1))))

(define accumulate-aux
 (lambda (amount n-years year)
       ...
```

---------- FIM da Prova Prática ----------