Programação em Python

Fundamentos e Resolução de Problemas

Ernesto Costa

Versão 0.96

SOLUÇÕES

Departamento de Engenharia Informática, Universidade de Coimbra
Polo II - Pinhal de Marrocos, 3030-290 Coimbra
http://ernesto.dei.uc.pt

Introdução

> If I had an hour to solve a problem I'd spend 55 minutes thinking about the problem and 5 minutes thinking about solutions.
>
> Albert Einstein

Apresentam-se neste texto algumas das soluções aos problemas do livro *Programação em Python: fundamentos e resolução de problemas*, publicado pela FCA. Adverte-se desde já o leitor que o que aqui apresentamos não são, em muitos casos, as soluções únicas. Também não são necessariamente as melhores, embora se tenha procurado que assim possam ser consideradas à luz dos conhecimentos anteriores ao capítulo onde aparecem enunciados.

Muitos destes destes exercícios não são originais, encontrando-se várias soluções para eles, seja em Python ou noutra linguagem, em diferentes textos. A sua utilidade é, em muitos casos, apenas didática. Muitos deles foram sendo propostos aos alunos da disciplina de **Introdução à Programação e Resolução de Problemas (IPRP)** do curso de Engenharia Informática da Universidade de Coimbra ao longo dos anos. Aos meus alunos e a todos os docentes que ao longo do ano me têm auxiliado nesta disciplina o meu agradecimento. Um agradecimento especial é devido àqueles que, mais recentemente e de modo mais continuado, me têm ajudado: Filipe Araújo, Tiago Baptista, Rui Lopes, Nuno Lourenço, Bernardete Ribeiro e João Vilela.

Existe um blogue associado à disciplina de IPRP, acessível através da ligação `http://programacaocompython.blogspot.com`. Aí o leitor encontrará muitos mais exemplos resolvidos, alguns conceitos explicados e outros elementos relevantes para a programação com Python.

Terei todo o gosto em receber a opinião dos leitores, seja na forma de correcções de eventuais erros, seja na forma de novas propostas. Posso ser contactado por correio electrónico para: ernesto@dei.uc.pt. Boas soluções!

Ernesto Costa

Parte I

Programação Procedimental

Capítulo 1

Introdução

Exercício 1.1 MF

Solução Windows: executar o IDLE (python gui) ou executar cmd; ir para o directório `c:\Python34` e executar o ficheiro python.exe
Linux e Mac OS: na linha de comandos executar o comando python3

Exercício 1.2 MF

Solução

```
def computacoes():
    """ Efectua diversas computações simples. """
    print("2+4 =",2+4)
    print("40*300 =",40*300)
    print("1/2 =",1/2)
    print("1.0/2 =",1.0/2)
    print("1.0//2 =",1.0//2)
    print("20e30*4 =",20e30*4)
    print("20e50*20e50 =",20e50*20e50)
    print("7%5 =",7%5)
    print("(5+2j)+(3+4j) =",(5+2j)+(3+4j))
    print("(5+2j)*(3+4j) =",(5+2j)*(3+4j))
    print("(5+2j)/(3+4j) =",(5+2j)/(3+4j))
```

Exercício 1.3 MF

Solução

```
""" Se usar em modo interactivo não necessita do print."""
print((9459 * 10**12) * (2.9 * 10**6))
```

Exercício 1.4 MF

Solução

```
""" Se usar em modo interactivo não necessita do print."""
print (365 * 24 * 60 * 60)
```

Exercício 1.5 MF Solução

```
""" Se usar em modo interactivo não necessita do print."""
print ((8*6) // (2*2))
```

Exercício 1.6 MF

Solução

```
def atributos_objectos():
    """ Mostra os atributos de três tipos numéricos diferentes
        . """
    int = 5
    print("Atributos de",int,":",id(int),type(int),int)
    float = 5.0
    print("Atributos de",float,":",id(float),type(float),float
        )
    imag = 2+5j
    print("Atributos de",imag,":",id(imag),type(imag),imag)
```

Exercício 1.7 MF

Solução

```
""" Caso concreto."""
base = 5
altura = 7

area = base * altura / 2
print(area)

"""Caso geral."""

def area(base, altura):
```

```
    return base * altura / 2

print(area(5,7))
```

Exercício 1.8 F

Solução

```
nova_area = 7.62 ** 2

velha_area = 3.1459 * (8.89/2)**2

print('nova= ', nova_area, 'velha= ', velha_area)
```

Exercício 1.9 F

Solução

```
def imc(peso, altura):
    return peso/altura**2
```

Exercício 1.10 F

Solução

```
def converte_c_to_f(celcius):
    return 9/5 * celcius + 32
```

Exercício 1.11 F

Solução

```
def volume_cone(raio, altura):
    return 3.1459 * (raio**2) * altura / 3
```

Exercício 1.12 F

Solução

```
def valor_poli(x):
    return x**4 + x**3 + 2*x**2 - x

print(valor_poli(1.1))
```

Exercício 1.13 F

Solução

```
import math
print(math.sin(5)) # --> -0.9589242746631385
print (math.sqrt(-4))
"""
Traceback (most recent call last):
  File "/Volumes/Work/LIVRO_PYTHON_2012/IPRP_LIVRO_2013_07/
     introd/programas/intro_sol.py", line 72, in <module>
builtins.ValueError: math domain error
"""
```

Exercício 1.14 F

Solução

```
def cambio_1(valor, taxa):
    return valor * taxa
```

Exercício 1.15 F

Solução

```
def numero_garrafas(quantidade_agua):
    """ Determina o número mínimo de garrafas de 5L, 1.5L, 0.5
       L e 0.25L
    que é necessário utilizar para albergar uma dada
       quantidade de água. """
    garrafas_5 = int(quantidade_agua//5)
    quantidade_agua = quantidade_agua%5
    garrafas_15 = int(quantidade_agua//1.5)
    quantidade_agua = quantidade_agua%1.5
    garrafas_05 = int(quantidade_agua//0.5)
    quantidade_agua = quantidade_agua%0.5
    garrafas_025 = math.ceil(quantidade_agua / 0.25)
    return garrafas_5 + garrafas_15 + garrafas_05 +
       garrafas_025
```

Exercício 1.16 M

Solução

```
import random

def jogar():
    tentativas = 3
    print("Bem vindo ao jogo da adivinha.")
    print("Tem %d tentativas para adivinhar um número inteiro
       entre 0 e 100" % tentativas)
    print("Eu ajudo...")
    jogar = eval(input("Vamos jogar? Entre 1 para Sim, e 0
       para Não:  "))
    acertou = 0
    total_tentativas = 0
    jogos = 0
    while jogar:
        tent, acer = adivinha(tentativas)
        total_tentativas = total_tentativas + tent
        acertou = acertou + acer
        jogos = jogos + 1
        jogar = eval(input("Vamos jogar mais? Entre 1 para Sim
           , e 0 para Não:  "))
    if jogos == 0:
        print("Então não quer jogar???")
        print("Até à próxima!")
    else:
        print("Até à próxima!")
        print("Número de jogos: %d" % jogos)
        print("Percentagem de acerto: %3.2f." % (acertou/jogos
           ))
        print("Número médio de tentativas: %3.2f." % (
           total_tentativas/jogos))
```

Exercício 1.17 Módulo math M

Solução

```
import math

def cartesianas_polares(x,y):
    """
    Converte coordenadas cartesianas em polares.
```

```
    Atenção ao uso de atan2 para dar o valor certo para todos
       os quadrantes.
    """
    r = math.sqrt(x**2+y**2)
    theta = math.atan2(y/x)
    return (r,theta)
```

Exercício 1.18 Módulo math **F**

Solução

```
def periodo_orbital_planeta(distancia_sol):
    """ Calcula o período, em anos, da órbita de um planeta,
       dada
    a sua distância ao Sol em Unidades Astronómicas (AU). """
    p = math.sqrt(distancia_sol**3)
    return p
```

Exercício 1.19 **M**

Solução

```
def periodo_orbital_newton(distancia, m1, m2):
    """ Calcula o período orbital (em dias) de qualquer corpo
       que orbita
    em volta de qualquer estrela, a partir da distância (AUs)
       entre os
    corpos e a massa solar de cada um deles (m1 e m2).
    Atenção às unidades!"""
    G = 6.67e-11 # m^3 kg^(-1) s^(-2)
    p = math.sqrt(4*math.pi**2*distancia**3/(G*(m1+m2))) #
       segundos
    return p/(24*60*60) # dias
```

Capítulo 2

Visões I

Exercício 2.1 F

Solução

```
def formas(num_lados, comp_lado, angulo_viragem):
    """
    Desenha formas regulares, fazendo variar os três
       parâmetros.
    >>> formas(4,100,144)
    desenha uma estrela.
    """
    for i in range(num_lados):
        turtle.forward(comp_lado)
        turtle.right(angulo_viragem)
    turtle.hideturtle()
```

Exercício 2.2 F

Solução

```
import turtle

def estrela_espiral(tarta,repete,incremento):
    """
    Desenha uma estrela em espiral.
    """
    for i in range(repete):
        tarta.forward(i * incremento)
        tarta.right(144)
```

```
if __name__ == '__main__':
    tarta = turtle.Turtle()
    estrela_espiral(tarta,30,8)
    turtle.exitonclick()
```

Exercício 2.3 F

Solução

```
import random

def random_walk(num_lados,comp_lado):
    """
    Simula um passeio aleatório.Muda a cor em cada etapa.
    """
    # Define modo de cor
    turtle.colormode(255)
    for i in range(num_lados):
        # define cor
        r = random.randint(0,255)
        g = random.randint(0,255)
        b = random.randint(0,255)
        turtle.pencolor((r,g,b))
        # Avança
        turtle.forward(comp_lado)
        # Vira
        turtle.setheading(random.randint(0,360))
    turtle.hideturtle()

def random_walk_2(num_lados,comp_lado):
    """
    Simula um passeio aleatório.Muda a cor em cada etapa.
    """
    for i in range(num_lados):
        # define cor
        r = random.random()
        g = random.random()
        b = random.random()
        turtle.pencolor((r,g,b))
        # Avança
```

```
        turtle.forward(comp_lado)
        # Vira
        turtle.setheading(random.randint(0,360))
    turtle.hideturtle()

# variante

def random_walk_var(num_lados,comp_lado):
    """
    Simula um passeio aleatório. Só três direcções: esquerda,
       direita, frente.
    """
    for i in range(num_lados):
        # Avança
        turtle.forward(comp_lado)
        # Vira
        dir = random.randint(0,2)
        if dir == 1:
            turtle.setheading(turtle.heading() + 90)
        elif dir == 2:
            turtle.setheading(turtle.heading() - 90)
    turtle.hideturtle()
```

Exercício 2.4 F

Solução

```
def poligono_regular(comp_lado,num_lados):
    """
    Desenha um polígono regular.
    """
    angulo_viragem = 360 / num_lados
    # Desenha
    for i in range(num_lados):
        turtle.forward(comp_lado)
        turtle.right(angulo_viragem)
    turtle.hideturtle()

import math
def circunferencia(raio):
    """
```

```
    Desenha uma circunferência conhecido o raio.
    """
    perimetro = 2 * math.pi * raio
    tam_lado = perimetro / 360
    poligono_regular(tam_lado,360)



def circunferencia_var(raio, x_cor, y_cor):
    """
    Desenha uma circunferência conhecido o raio. Centro numa
       posição definida.
    """
    turtle.penup()
    turtle.goto(x_cor, y_cor)
    turtle.pendown()
    perimetro = 2 * math.pi * raio
    tam_lado = perimetro / 360
    poligono_regular(tam_lado,360)
```

Exercício 2.5 F

Solução

```
import turtle

def forma(nome,pontos):
    nova_forma = turtle.Shape('polygon',pontos)
    turtle.register_shape(nome,nova_forma)

if __name__ == '__main__':
    nome = 'oops'
    forma(nome,((0,0),(10,20),(5,20),(0,30),(-5,20),(-10,20)))
    turtle.shape(nome)
    turtle.fillcolor('red')
    turtle.forward(100)
    turtle.right(45)
    turtle.forward(50)
    turtle.exitonclick()
```

Exercício 2.6 M

Solução

```
def poly_circle(num_lados, comp_lado):
    """
    Desenha um polígono regular usando o comando circle.
    """
    turtle.circle(comp_lado,360,num_lados)
    turtle.hideturtle()

def poly_circle_var(num_lados, comp_lado):
    """
    Desenha um polígono regular usando o comando circle.
    """
    turtle.fillcolor('orange')
    turtle.begin_fill()
    turtle.width(3)
    turtle.circle(comp_lado,360,num_lados)
    turtle.end_fill()
    turtle.hideturtle()
```

Exercício 2.7 M

Solução

```
import turtle

def smiley():
    circunferencia(0,0,100,'white')
    circunferencia(-40,125,15,'black')
    circunferencia(40,125,15,'black')
    turtle.right(30)
    circunferencia(-55,50,100,'white',65)

def circunferencia(pos_x, pos_y,raio,cor,angulo=360):
    """pos_xz e pos_y são o centro!"""
    turtle.penup()
    turtle.goto(pos_x ,pos_y)
    turtle.fillcolor(cor)
    turtle.pendown()
    turtle.showturtle()
    turtle.begin_fill()
    turtle.circle(raio,angulo)
```

```
    turtle.end_fill()

    print(turtle.position())
    turtle.hideturtle()
```

Exercício 2.8 M

Solução

```
import turtle

def avanca(tarta,dist, raio):
    """
    Mantém a tartaruga prisioneira num raio da origem.
    """
    x_cor = tarta.xcor()
    y_cor = tarta.ycor()
    for i in range(dist):
        if tarta.distance(x_cor,y_cor) > raio:
            angulo = tarta.towards(x_cor,y_cor)
            tarta.setheading(angulo)
        tarta.forward(1)


import random

def avanca_2(tarta,dist, raio):
    """
    Mantém a tartaruga prisioneira num raio da origem.
    """
    x_cor = tarta.xcor()
    y_cor = tarta.ycor()
    for i in range(dist):
        if tarta.distance(x_cor,y_cor) > raio:
            angulo = tarta.towards(x_cor + random.randint
               (-25,25),x_cor + random.randint(-5,5))
            tarta.setheading(angulo)
        tarta.forward(1)
```

```
tarta = turtle.Turtle()
avanca_2(tarta,10000,150)
turtle.exitonclick()
```

Exercício 2.9 M

Solução

```
import turtle

def rasto(tarta, tamanho, passo, angulo, repete,cor_1, cor_2):
   janela = turtle.Screen()
   janela.bgcolor(cor_1)

   tarta.color(cor_2)

   tarta.penup()
   for i in range(repete):
      tarta.stamp()
      tamanho = tamanho + passo
      tarta.forward(tamanho)
      tarta.right(angulo)

   janela.mainloop()

if __name__ == '__main__':
   tarta = turtle.Turtle()
   tarta.shape("turtle")
   rasto(tarta,20,3,24,30,'blue','red')
   turtle.exitonclick()
```

Exercício 2.10 M

Solução

```
import turtle

def relogio(tarta, raio,cor_1, cor_2):
   janela = turtle.Screen()
   janela.bgcolor(cor_1)
   janela.title('Relógio')
   tarta.color(cor_2)
```

```
   tarta.penup()
   tarta.setheading(60)
   for i in range(12):
      tarta.forward(raio)
      tarta.pendown()
      tarta.forward(30)
      tarta.penup()
      tarta.forward(30)
      tarta.pendown()
      tarta.write(i + 1, align='center', font=('Arial',20,'
         bold'))
      tarta.penup()
      tarta.goto(0,0)
      tarta.right(30)
   tarta.hideturtle()


   janela.mainloop()

if __name__ == '__main__':
   tarta = turtle.Turtle()
   #tarta.shape("turtle")
   relogio(tarta,150,'lightgreen','red')
   turtle.exitonclick()
```

Exercício 2.11 M

Solução

```
import turtle

def triangulo(posx,posy,lado, orientacao):
    turtle.penup()
    turtle.goto(posx,posy)
    turtle.pendown()
    turtle.setheading(orientacao)
    for i in range(3):
        turtle.forward(lado)
        turtle.left(120)

def main(posx,posy,lado, orientacao):
```

```
    for i in range(6):
        triangulo(posx,posy,lado,orientacao)
        orientacao = orientacao + 60

if __name__ == '__main__':
    main(0,0,100,60)
    turtle.hideturtle()
    turtle.exitonclick()
```

Exercício 2.12 M

Solução

```
import turtle

def triangulo(posx,posy,lado, orientacao):
    turtle.penup()
    turtle.goto(posx,posy)
    turtle.pendown()
    turtle.setheading(orientacao)
    for i in range(3):
        turtle.forward(lado)
        turtle.left(120)

def rectangulo(posx,posy,lado1, lado2, orientacao):
    turtle.penup()
    turtle.goto(posx,posy)
    turtle.pendown()
    turtle.setheading(orientacao)
    for i in range(2):
        turtle.forward(lado1)
        turtle.left(90)
        turtle.forward(lado2)
        turtle.left(90)

if __name__ == '__main__':
    lado1 = 100
    lado2 = 30
    turtle.fillcolor('green')
    turtle.begin_fill()
    rectangulo(0,0, lado1,lado1,0)
```

```
    turtle.end_fill()
    turtle.fillcolor('black')
    turtle.begin_fill()
    rectangulo(1.9*lado1/3, 6*lado1/4,lado2/4,lado1/4,0)
    turtle.end_fill()
    turtle.fillcolor('red')
    turtle.begin_fill()
    triangulo(0,lado1,lado1,0)
    turtle.end_fill()

    turtle.hideturtle()
    turtle.exitonclick()
```

Exercício 2.13 M

Solução

```
import turtle

def esse(raio):
    turtle.setheading(270)
    turtle.circle(raio,180)
    turtle.setheading(270)
    turtle.circle(raio,-180)
    turtle.hideturtle()


if __name__ == '__main__':
    esse(30)
```

Exercício 2.14 D

Solução

```
def quadrados_concentricos(num,comp_lado,distancia):
    """
    Desenha num quadrados concentricos.
    """
    x = turtle.xcor()
    y = turtle.ycor()
    for i in range(num):
  # Define cor
```

```
        r = random.random()
        g = random.random()
        b = random.random()
  # Desenha quadrado
        quadrado_cor(comp_lado+i*2*distancia, x - i*distancia,
            y + i*distancia,0,(r,g,b))


def quadrado_cor(lado,xcor,ycor,orient, cor):
    """
    Desenha um quadrado em que o lado, a posição inicial, a
       orientação inicial e a cor podem variar.
    """
    turtle.penup()
    turtle.goto(xcor,ycor)
    turtle.setheading(orient)
    turtle.pencolor(cor)
    turtle.pendown()
    for i in range(4):
        turtle.forward(lado)
        turtle.right(90)
    turtle.hideturtle()

def quadrado(lado,xcor,ycor,orient):
    """
    Desenha um quadrado em que o lado, a posição inicial e a
       orientação inicial podem variar.
    """
    turtle.penup()
    turtle.goto(xcor,ycor)
    turtle.setheading(orient)
    turtle.pendown()
    for i in range(4):
        turtle.forward(lado)
        turtle.right(90)
    turtle.hideturtle()

# variante

def quadrados_concentricos_var(num,comp_lado,distancia,
   orienta):
```

```
    """
    Desenha num quadrados concentricos.
    """
    x = turtle.xcor()
    y = turtle.ycor()
    for i in range(num):
  # Define cor
        r = random.random()
        g = random.random()
        b = random.random()
  # Desenha quadrado
        quadrado_cor(comp_lado+i*2*distancia, x - i*distancia,
            y + i*distancia,orienta * i,(r,g,b))
```

Exercício 2.15 D

Solução

```
def nautilus(n, lado, xcor,ycor, angulo):
    """
    Desenha n quadrados com lados e orientações variáveis.
    Desenha uma forma semelhante a um Nautilus.

    """
    for conta in range(n):
        quadrado(lado,xcor,ycor,angulo)
        lado = lado + 10
        angulo = angulo + 15
    turtle.hideturtle()
```

Exercício 2.16 M

Solução

```
def circunferencia_cor(x,y,raio,cor):
    """
    Desenha circunferência colorida.
    """
    turtle.penup()
    turtle.setpos(x,y-raio)
    turtle.pendown()
    turtle.color(cor)
```

```
    turtle.circle(raio)
    turtle.penup()

def figura(raio,delta):
    """
    O símbolo dos jogos olímpicos, colorido.
    """
    turtle.width(5)
    turtle.colormode(255)
    circunferencia_cor(0,0,raio,'black')
    circunferencia_cor(2*raio+delta,0,raio,'red')
    circunferencia_cor(-2*raio-delta,0,raio,'blue')
    circunferencia_cor(raio+delta/2,-raio,raio,'green')
    circunferencia_cor(-raio-delta/2,-raio,raio,'orange')
    turtle.hideturtle()
```

Exercício 2.17 D

Solução

```
def radioactividade(lado,x_cor,y_cor):
    fundo(lado,x_cor,y_cor,'yellow')
    vai_para(x_cor, y_cor)
    tres_sectores(0.8*lado/2)
    raio_2 = 0.2*lado/2
    circulo(raio_2,x_cor,y_cor,'black','yellow')
    turtle.hideturtle()

def vai_para(x,y):
    turtle.penup()
    turtle.goto(x,y)
    turtle.pendown()

def quad(lado):
    for i in range(4):
        turtle.forward(lado)
        turtle.left(90)

def sector(raio, angulo,cor, orient):
    # Define orientação
    turtle.setheading(orient)
```

```
    # define cor
    turtle.fillcolor(cor)
    turtle.pencolor(cor)
    # Desenha
    turtle.begin_fill()
    turtle.forward(raio)
    turtle.left(90)
    turtle.circle(raio, angulo)
    turtle.left(90)
    turtle.forward(raio)
    turtle.left(180-angulo)
    turtle.end_fill()

def tres_sectores(raio):
    """
    Desenha três sectores de círculo igualmente espaçados e
       com fundo preto.
    """
    sector(raio,60,'black',0)
    sector(raio,60,'black',120)
    sector(raio,60,'black',240)

def fundo(lado,x_cor,y_cor,cor):
    # Popsiciona
    turtle.penup()
    turtle.goto(x_cor - lado/2,y_cor - lado/2)
    turtle.pendown()
    # Prepara
    turtle.width(3)
    turtle.fillcolor(cor)
    turtle.begin_fill()
    # Desenha
    quad(lado)
    # Termina
    turtle.end_fill()

def circulo(raio,x_cor,y_cor,cor_1,cor_2):
    # Prepara
    vai_para(x_cor,y_cor+raio)
    turtle.right(60)
    turtle.fillcolor(cor_1)
```

```
    turtle.pencolor(cor_2)
    turtle.width(3)
    turtle.begin_fill()
    # Desenha
    turtle.circle(raio)
    turtle.end_fill()
```

Exercício 2.18 D

Solução

```
def zen(raio,x_cor,y_cor,orient, cor_1,cor_2):
    """
    Desenha parte do símbolo zen.
    """
    vai_para(x_cor,y_cor)
    turtle.setheading(orient)
    turtle.pencolor(cor_1)
    turtle.fillcolor(cor_2)
    turtle.begin_fill()
    turtle.circle(raio,180)
    turtle.circle(raio/2,-180)
    turtle.setheading(orient + 180)
    turtle.circle(raio/2,180)
    turtle.end_fill()

def yin_yang(raio):
    zen(raio,0,0,90,'black','white')
    zen(raio,-200,0,270,'black','black')
    circulo(raio/10,-55,0,'black','black')
    circulo(raio/10,-145,0,'white','white')
    turtle.hideturtle()
```

Exercício 2.19 D

Solução

```
import turtle

def grelha(tarta,dist_pontos, largura, altura):
    tarta.penup()
    tarta.hideturtle()
```

```
    for y in range(altura):
        for i in range(largura):
            tarta.pensize(5)
            tarta.dot()
            tarta.forward(dist_pontos)
        tarta.backward(dist_pontos * largura)
        tarta.left(90)
        tarta.forward(dist_pontos)
        tarta.right(90)

def jogo_do_galo():
    tarta_O = turtle.Turtle()
    tarta_O.hideturtle()
    tarta_O.penup()
    tarta_O.pencolor('red')
    tarta_O.pensize(5)

    tarta_X= turtle.Turtle()
    tarta_X.hideturtle()
    tarta_X.penup()
    tarta_X.pencolor('green')
    tarta_X.pensize(5)

    jogador = tarta_O
    nome = '_O_'
    for i in range(9):
        pos_x , pos_y = eval(input('Coordenadas  (X, Y) para o
            jogador %s: ' % nome))
        jogador.goto(pos_x * 40, pos_y * 40)
        jogador.dot()
        if jogador == tarta_O:
            jogador = tarta_X
            nome = '_X_'
        else:
            jogador = tarta_O
            nome = '_O_'

if __name__ == '__main__':
    tarta = turtle.Turtle()
    #tarta_o = turtle.Turtle()
    #tarta_x = turtle.Turtle()
```

```
    dist_pontos = 40
    largura = 3
    altura =  3
    grelha(tarta, dist_pontos,altura, largura)
    jogo_do_galo()

    turtle.exitonclick()
```

Capítulo 3

Objectos (I)

Exercício 3.1 MF

Solução

```
Python 3.2.3 (default, Sep  5 2012, 20:52:27)
[GCC 4.2.1 (Based on Apple Inc. build 5658) (LLVM build
   2336.1.00)]
Type "help", "copyright", "credits" or "license" for more
   information.
a = 5
help(a)
Help on int object:

class int(object)
 |  int(x[, base]) -> integer
 |
 |  Convert a string or number to an integer, if possible.  A
    floating
 |  point argument will be truncated towards zero (this does
    not include a
 |  string representation of a floating point number!)  When
    converting a
 |  string, use the optional base.  It is an error to supply a
     base when
 |  converting a non-string.
 |
 |  Methods defined here:
 |
```

```
|  __abs__(...)
|      x.__abs__() <==> abs(x)
|
|  __add__(...)
|      x.__add__(y) <==> x+y
|
|  __and__(...)
|      x.__and__(y) <==> x&y
|
|  __bool__(...)
|      x.__bool__() <==> x != 0
|
|  __ceil__(...)
|      Ceiling of an Integral returns itself.
|
|  __divmod__(...)
|      x.__divmod__(y) <==> divmod(x, y)
|
|  __eq__(...)
|      x.__eq__(y) <==> x==y
|
|  __float__(...)
|      x.__float__() <==> float(x)
|
|  __floor__(...)
|      Flooring an Integral returns itself.
|
|  __floordiv__(...)
|      x.__floordiv__(y) <==> x//y
|
|  __format__(...)
|
|  __ge__(...)
|      x.__ge__(y) <==> x>=y
|
|  __getattribute__(...)
|      x.__getattribute__('name') <==> x.name
|
|  __getnewargs__(...)
|
|  __gt__(...)
```

```
|      x.__gt__(y) <==> x>y
|
|  __hash__(...)
|      x.__hash__() <==> hash(x)
|
|  __index__(...)
|      x[y:z] <==> x[y.__index__():z.__index__()]
|
|  __int__(...)
|      x.__int__() <==> int(x)
|
|  __invert__(...)
|      x.__invert__() <==> ~x
|
|  __le__(...)
|      x.__le__(y) <==> x<=y
|
|  __lshift__(...)
|      x.__lshift__(y) <==> x<<y
|
|  __lt__(...)
|      x.__lt__(y) <==> x<y
|
|  __mod__(...)
|      x.__mod__(y) <==> x%y
|
|  __mul__(...)
|      x.__mul__(y) <==> x*y
|
|  __ne__(...)
|      x.__ne__(y) <==> x!=y
|
|  __neg__(...)
|      x.__neg__() <==> -x
|
|  __or__(...)
|      x.__or__(y) <==> x|y
|
|  __pos__(...)
|      x.__pos__() <==> +x
|
```

```
|  __pow__(...)
|      x.__pow__(y[, z]) <==> pow(x, y[, z])
|
|  __radd__(...)
|      x.__radd__(y) <==> y+x
|
|  __rand__(...)
|      x.__rand__(y) <==> y&x
|
|  __rdivmod__(...)
|      x.__rdivmod__(y) <==> divmod(y, x)
|
|  __repr__(...)
|      x.__repr__() <==> repr(x)
|
|  __rfloordiv__(...)
|      x.__rfloordiv__(y) <==> y//x
|
|  __rlshift__(...)
|      x.__rlshift__(y) <==> y<<x
|
|  __rmod__(...)
|      x.__rmod__(y) <==> y%x
|
|  __rmul__(...)
|      x.__rmul__(y) <==> y*x
|
|  __ror__(...)
|      x.__ror__(y) <==> y|x
|
|  __round__(...)
|      Rounding an Integral returns itself.
|      Rounding with an ndigits argument also returns an
   integer.
|
|  __rpow__(...)
|      y.__rpow__(x[, z]) <==> pow(x, y[, z])
|
|  __rrshift__(...)
|      x.__rrshift__(y) <==> y>>x
|
```

```
|  __rshift__(...)
|      x.__rshift__(y) <==> x>>y
|
|  __rsub__(...)
|      x.__rsub__(y) <==> y-x
|
|  __rtruediv__(...)
|      x.__rtruediv__(y) <==> y/x
|
|  __rxor__(...)
|      x.__rxor__(y) <==> y^x
|
|  __sizeof__(...)
|      Returns size in memory, in bytes
|
|  __str__(...)
|      x.__str__() <==> str(x)
|
|  __sub__(...)
|      x.__sub__(y) <==> x-y
|
|  __truediv__(...)
|      x.__truediv__(y) <==> x/y
|
|  __trunc__(...)
|      Truncating an Integral returns itself.
|
|  __xor__(...)
|      x.__xor__(y) <==> x^y
|
|  bit_length(...)
|      int.bit_length() -> int
|
|      Number of bits necessary to represent self in binary.
|      >>> bin(37)
|      '0b100101'
|      >>> (37).bit_length()
|      6
|
|  conjugate(...)
|      Returns self, the complex conjugate of any int.
```

```
|
|  from_bytes(...)
|      int.from_bytes(bytes, byteorder, *, signed=False) ->
   int
|
|      Return the integer represented by the given array of
   bytes.
|
|      The bytes argument must either support the buffer
   protocol or be an
|      iterable object producing bytes.  Bytes and bytearray
   are examples of
|      built-in objects that support the buffer protocol.
|
|      The byteorder argument determines the byte order used
   to represent the
|      integer.  If byteorder is 'big', the most significant
   byte is at the
|      beginning of the byte array.  If byteorder is 'little'
   , the most
|      significant byte is at the end of the byte array.  To
   request the native
|      byte order of the host system, use `sys.byteorder' as
   the byte order value.
|
|      The signed keyword-only argument indicates whether two
   's complement is
|      used to represent the integer.
|
|  to_bytes(...)
|      int.to_bytes(length, byteorder, *, signed=False) ->
   bytes
|
|      Return an array of bytes representing an integer.
|
|      The integer is represented using length bytes.  An
   OverflowError is
|      raised if the integer is not representable with the
   given number of
|      bytes.
|
```

```
|      The byteorder argument determines the byte order used
   to represent the
|      integer.  If byteorder is 'big', the most significant
   byte is at the
|      beginning of the byte array.  If byteorder is 'little'
   , the most
|      significant byte is at the end of the byte array.  To
   request the native
|      byte order of the host system, use `sys.byteorder' as
   the byte order value.
|
|      The signed keyword-only argument determines whether
   two's complement is
|      used to represent the integer.  If signed is False and
    a negative integer
|      is given, an OverflowError is raised.
|
|
   ----------------------------------------------------------------------

|  Data descriptors defined here:
|
|  denominator
|      the denominator of a rational number in lowest terms
|
|  imag
|      the imaginary part of a complex number
|
|  numerator
|      the numerator of a rational number in lowest terms
|
|  real
|      the real part of a complex number
|
|
   ----------------------------------------------------------------------

|  Data and other attributes defined here:
|
|  __new__ = <built-in method __new__ of type object>
|      T.__new__(S, ...) -> a new object with type S, a
```

```
    subtype of T
```

A informação mostra as características do **tipo** inteiro, incluindo o seu construtor e os diferentes métodos. Estes aparecem na formulação de métodos especiais.

Exercício 3.2 F

Solução

```
import math

def area_tri(a,b,c):
    """Area do triangulo: método de Heron."""
    s = (a + b + c) / 2
    return math.sqrt(s * (s - a) * (s - b) * (s - c))
```

Exercício 3.3 F

Solução

```
import math

def escada(altura, angulo):
    """
    Tamanho de uma escada para ser encostada a um parede com
       uma dada altura
    e fazendo um certoangulo.
    """
    return altura / math.sin(angulo)

def escada_graus(altura, angulo):
    """
    Tamanho de uma escada para ser encostada a um parede com
       uma dada altura
    e fazendo um certo angulo dado em graus.
    """
    radianos = math.pi/ 180 * angulo
    return altura / math.sin(radianos)
```

Exercício 3.4 F

Solução

```
def bate_card_max(idade):
    """O batimento cardíaco aproximado em função da idade."""
    return 163 + 1.16 * idade - 0.018 * (idade **2)
```

Exercício 3.5 F

Solução

```
def ganhos(inicial, taxa_juro, tempo):
    """
    Taxa de juro composto. Um período de capitalização
    """
    return inicial * (1 + taxa_juro)** tempo
```

Exercício 3.6 F

Solução

```
def ganhos_per(inicial, taxa_juro, tempo,per):
    """
    Taxa de juro composto. Vários períodos de capitalização
    """
    return inicial * (1 + taxa_juro/per)** (tempo*per)
```

Capitalizar com vários períodos é melhor.

Exercício 3.7 M

Solução

```
import math

def entropia(n,m):
    if (n == 0) or (m == 0):
        return 0
    p_1 = n / (n+m)
    p_2 = m / (n+m)
    return - p_1 * math.log2(p_1) - p_2 * math.log2(p_2)
```

Exercício 3.8 M

Solução

```
def biseccao(func,a,b,n):
```

```
    """
    raízes de um polinómio pelo método da bisecção.
    função contínua em [a,b]
    a < b
    f(a) * f(b) < 0
    """
    for i in range(n):
        x = (a + b)/ 2
        res = func(x)
        print(x)
        if res * func(b) < 0:
            a = x
        else:
            b = x

def biseccao_b(func,a,b,eps):
    """
    raízes de um polinómio pelo método da bisecção.
    função contínua em [a,b]
    a < b
    f(a) * f(b) < 0
    """
    while True:
        x = (a + b)/ 2
        res = func(x)
        print(x)
        if abs(res) < eps:
            break
        if res < 0:
            a = x
        else:
            b = x

def poli_3(x):
    return x**3 - x - 1

if __name__ == '__main__':
    biseccao(poli_3,0,2,20)
    biseccao_b(poli_3,0,2,0.003)
```

Exercício 3.9 F

Solução

```
def desencripta(texto_encriptado):
    """Desencriptação por separação dos caracteres nas
       posições pares
    e nas posições ímpares. As posições ímpares foram
       colocadas primeiro!"""
    comp = len(texto_encriptado)
    meio = comp // 2
    # pares
    car_pares = texto_encriptado[meio:]
    # impares
    car_impares = texto_encriptado[:meio]
    # junta
    texto_normal = ''
    for i in range(meio):
        texto_normal = texto_normal + car_pares[i] +
           car_impares[i]
    if comp % 2 != 0:
        texto_normal = texto_normal + car_pares[-1]# tamanho
           ímpar
    return texto_normal
```

Exercício 3.10 F

Solução

```
def descodifica_1(texto_encriptado,chave):
    """Descodifica um texto pelo método de substituição. A
       chave é
    dada por uma correspondência um a um entre caracteres.
       Supõe que
    os caracteres são as 26 letras (minúsculas) do alfabeto
       mais o espaço em branco"""
    alfabeto = 'abcdefghijklmnopqrstuvwxyz '
    texto_normal = ''
    for car in texto_encriptado:
        indice = chave.find(car)
        texto_normal = texto_normal + alfabeto[indice]
```

```
    return texto_normal
```

Exercício 3.11 F

Solução

```
def complemento(adn):
    """Calcula o complemento de uma cadeia de ADN usando
       condicionais."""
    comp = ''
    for i in range(len(adn)):
        if adn[i] == 'A':
            comp = comp + 'T'
        elif adn[i] == 'T':
            comp = comp + 'A'
        elif adn[i] == 'C':
            comp = comp + 'G'
        elif adn[i] == 'G':
            comp = comp + 'C'
        else:
            print('ERRO: símbolo desconhecido!')
            return False
    return comp
```

Exercício 3.12 F

Solução

```
def gera_adn_b(tam):
    """Gera uma cadeia de ADN de tamanho tam. Padrão ciclo-
       acumulador"""
    adn =''
    for i in range(tam):
        base = random.choice('TACG')
        adn = adn + base
    return adn

# variante para os pitónicos
def gera_adn(tam):
    """Gera uma cadeia de ADN de tamanho tam."""
    return ''.join([random.choice('TACG') for i in range(tam)
       ])
```

Exercício 3.13 F

Solução

```
def tira_vogais(cadeia):
    """Retira as vogais e substitui por um espaço em branco.
    """
    vogais ='aeiou'
    nova_cadeia =''
    for conta in range(len(cadeia)):
        if cadeia[conta] in vogais:
            nova_cadeia = nova_cadeia + ' '
        else:
            nova_cadeia = nova_cadeia + cadeia[conta]
    return nova_cadeia

def tira_vogais_b(cadeia):
    """Retira as vogais e substitui por um espaço em branco.
    """
    vogais ='aeiou'
    nova_cadeia =''
    for car in cadeia:
        if car in vogais:
            nova_cadeia = nova_cadeia + ' '
        else:
            nova_cadeia = nova_cadeia + car
    return nova_cadeia
```

Exercício 3.14 F

Solução

```
def sub_cadeias(pal, n):
    """
    Todas as subcadeias de comprimento n.
    """
    for i in range(len(pal) - n + 1):
        print(pal[i:i + n])
```

Exercício 3.15 M

Solução

```
def prefixos(pal):
    """
    Mostra os prefixos da palavra.
    """
    for i in range(len(pal)):
        print(pal[:i+1])
```

Exercício 3.16 M

Solução

```
def monty_suf():
    nome = 'Monty Python'
    for i in range(len(nome),-1,-1):
        print(nome[i:])

# Versão geral
def sufixos(pal):
    """
    Mostra os sufixos da palavra.
    """
    for i in range(len(pal),-1,-1):
        print(pal[i:])
```

Exercício 3.17 Módulo turtle F

Solução

```
import turtle

def adn_tartaruga(tartaruga, adn):
    """ Simula o comportamento da tartaruga ditado pelo seu
       ADN."""
    tartaruga.down()
    for car in adn:
        if car == 'f':
            tartaruga.fd(50)
        elif car == 't':
            tartaruga.bk(50)
        elif car == 'd':
```

```
            tartaruga.rt(45)
        else:
            tartaruga.lt(45)
```

Exercício 3.18 Módulo random Módulo turtle F

Solução

```
import random

def adn_tartaruga_alea(tartaruga, adn):
    """ Simula o comportamento da tartaruga ditado pelo seu
       ADN."""
    tartaruga.down()
    for car in adn:
        lado = random.randint(20,100)
        angulo = random.randint(10,180)
        if car == 'f':
            tartaruga.fd(lado)
        elif car == 't':
            tartaruga.bk(lado)
        elif car == 'd':
            tartaruga.rt(angulo)
        else:
            tartaruga.lt(angulo)
```

Exercício 3.19 Módulo random Módulo turtle F

Solução

```
def adn_tartaruga_total(tartaruga, passos):
    """ Simula o comportamento da tartaruga em função do seu
       ADN. O dito
    ADN é gerado aleatoriamente."""
    adn =''
    for i in range(passos):
        adn = adn + random.choice('fted')
    adn_tartaruga_alea(tartaruga,adn)
```

Exercício 3.20 M

Solução

```
def codifica(texto_normal,chave):
    """Codifica um texto pelo método de substituição. A chave
       é a distância
    para codificar. Exemplo: 'a' passa a 'c' se chave for 2.
       Supõe que
    os caracteres são as 26 letras (minúsculas) do alfabeto  e
        o branco."""
    alfabeto = 'abcdefghijklmnopqrstuvwxyz '
    texto_encriptado = ''
    for car in texto_normal:
        novo_codigo = (alfabeto.index(car) + chave) % len(
           alfabeto)
        novo_car = alfabeto[novo_codigo]
        texto_encriptado = texto_encriptado + novo_car
    return texto_encriptado


def descodifica(texto_encriptado,chave):
    """Descodifica um texto pelo método de substituição. A
       chave é
    dada por uma correspondência um a um entre caracteres.
       Supõe que
    os caracteres são as 26 letras (minúsculas) do alfabeto
       mais o espaço em branco"""
    alfabeto = 'abcdefghijklmnopqrstuvwxyz '
    texto_normal = ''
    for car in texto_encriptado:
        indice = alfabeto.find(car)
        texto_normal = texto_normal + alfabeto[(indice - chave
           )%len(alfabeto)]
    return texto_normal
```

Capítulo 4

Instruções destrutivas

Exercício 4.1 MF

Solução

Não são válidos:
segundo: começa por um digito.
quarto: existe um espaço em branco.
sexto: é uma palavra reservada.
sétimo: termina por um caractere não autorizado.
nono exemplo: palavra reservada.
décimo primeiro: uso de parênteses.
décimo quinto exemplo: uso do hífen.

Exercício 4.2 MF

Solução

```
"""
>>> chr(0x3B1)
'?'
>>>
"""
```

Exercício 4.3 MF

Solução

```
"""
>>> x = 5
>>> y = 5
```

```
>>> id(x)
4446072256
>>> 4446072256

Trata-se do mesmo objecto com dois nomes diferentes. Os
   inteiros pequenos
estão internalizados.
"""
```

Exercício 4.4 MF

Solução

```
"""
Semelhante ao caso anterior.
"""
```

Exercício 4.5 MF

Solução

```
""" A identidade alterou-se. O nome é um atributo do objecto.
A instrução de atribuição tem à esquerda um nome e à direita
   uma expressão. A expressão é avaliada e o objecto
encontrado é associado com o nome 'x', desfazendo a ligação
   anterior.
"""
```

Exercício 4.6 MF

Solução

```
"""
Junta os dois casos anteriores.
"""
```

Exercício 4.7 M

Solução

```
"""
Fazer o desenho...
"""
```

Exercício 4.8 M

Solução

```
"""
É feita uma cópia de superfície da cadeia 'a' com tamanho 3.
"""
```

Exercício 4.9 F

Solução

No primeiro caso o resultado é 46.8 que corresponde à soma de 23.4 consigo próprio. No segundo caso o resultado é a cadeira de caracteres `totototo`, que resulta de concatenar toto consigo próprio. Estamos perante a manifestação de sobrecarregamento do operador +.

Exercício 4.10 M

Solução

Começamos por definir a função `prod`. De seguida associamos o nome **a** ao objecto 5. Na terceira situação, mandamos imprimir o resultado de executar a função prod com **x** associado a **a** e **y** associado a 3. A execução da função corresponde a efectuar o produto de x por y, ou seja a vezes 3, pelo que o resultado é o esperado: 15. A linha seguinte mostra que a associação de **a** a 5 não foi alterada. Finalmente, depois de executada a função as ligações entre os parâmetros formais e os reais são desfeitas pelo que o nome **x** não existe fora da definição, originando então a mensagem de erro.

Exercício 4.11 F

Solução

```
"""
print('{0:20s}'.format('Bem vindo a IPRP'))
print('{0:>20s}'.format('Bem vindo a IPRP'))
print('{0:^40s}'.format('Bem vindo a IPRP e ao DEIUC'))
"""
```

Exercício 4.12 M

Solução

```
def imprime_tabela(numero):
    """ Tabela com os valores de numero, numero ^ 2."""
    print('Número\t\tQuadrado')
    for i in range(1,numero+1):
        print('%6d\t\t%8d' % (i, i**2))
```

Exercício 4.13 F

Solução

```
def converte_m_k(inf, sup):
    const = 1.609
    print('Milhas\t\tQuilómetros')
    print('-' * 28)
    for milhas in range(inf, sup+1):
        print("%5.2f\t\t%5.2f"% (milhas,milhas*const))

if __name__ == '__main__':
    converte_m_k(101,120)
```

Exercício 4.14 M

Solução

```
def tabuada(n):
    """Imprime a tabela da tabuada do númeron."""
    print('Tabuada do número  %d' % n)
    print('-'*20)
    for i in range(1,11):
        print('%d\tx\t%4d\t=%4d' % (n,i,i*n))
```

Exercício 4.15 M

Solução

```
def acronimo_alunos(cadeia):
    """Extrai o acrónimo da cadeia."""
    cadeia = cadeia.upper()
    acro = cadeia[0]
    for i in range(1, len(cadeia)-1):
        if cadeia[i] == ' ':
            acro = acro + cadeia[i+1]
    return acro
```

```
def acronimo(cadeia):
    acro = ''
    nova_cadeia = cadeia.strip().split()
    for pal in nova_cadeia:
        acro += pal[0].upper()
    return acro


def acronimo_b(cadeia):
    """
    Constrói o acrónimo a partir de uma frase
    representada por uma cadeia de caracteres.
    """
    acro = ''
    inicio = True
    for car in cadeia:
        if car == ' ':
            inicio = True
        elif inicio == True:
            acro += car.upper()
            inicio = False
    return acro

def acronimo_c(frase):
    """ Forma um acronimo a partir da frase."""
    frase = frase.upper().strip()
    comprimento = len(frase)
    acron = ''
    posicao = 0
    while posicao < comprimento:
        acron = acron + frase[posicao].upper()
        while (posicao < comprimento) and (frase[posicao] != '
            '):
            posicao = posicao + 1
        while (posicao < comprimento) and (frase[posicao] == '
            '):
            posicao = posicao + 1
    return acron
```

Exercício 4.16 F

Solução

```
def comprimento():
    """Calcula o tamanho da pista necessário para a descolagem
       ."""
    vel = eval(input('Velocidade de descolagem (m/s): '))
    ace = eval(input('Aceleração para descolagem (m/s?): '))
    comp = (vel**2) / (2*ace)
    print('Para a velocidade %3.2f e aceleração %3.2f o
       comprimento mínimo da pista é: %5.2f.' % (vel,ace,comp)
       )
```

Exercício 4.17 F

Solução

```
def energia():
    """Calcula o valor da energia necessária para variar a
       temperatura da água."""
    t_i = eval(input('Temperatura inicial  (Celsius): '))
    t_f = eval(input('Temperatura final (Celsius): '))
    m = eval(input('Quantidade de água (Quilogramas): '))

    e = m * (t_f - t_i) * 4184

    print('Para a massa de água %3.2f, temperatura inicial
       %3.2f e temperatura final %3.2f a energia necessária é:
        %10.2f Joules.' % (m,t_i,t_f,e))
```

Exercício 4.18 F

Solução

```
def temperatura():
    """Calcula o valor da temperatura exterior em função do
       vento."""
    vel = eval(input('Velocidade do vento (milhas/hora): '))
    temp = eval(input('Temperatura (Fahrenheit [-58, 41]): '))
```

```
    res = 35.74 + 0.6215 * temp - 35.75 * (vel**0.16) + 0.4275
        * temp * (vel ** 0.16)

    print('Para a velocidade do vento %3.2f e temperatura
       exterior %3.2f a temperatura é sentida conmo: %5.2f.' %
        (vel,temp,res))
```

Exercício 4.19 M

Solução

```
def mostra_matriz(matriz):
    """imprime os elementos de uma matriz de modo organizado.
       """
    print()
    for j,linha in enumerate(matriz):
        for i,coluna in enumerate(linha):
            print('(%d,%d): %d\t' % (j,i,coluna), end='')
        print()
```

Exercício 4.20 M

Solução

```
def estima(pop):
    """Ao fim de um ano qual o novo valor da população."""
    nasce = eval(input('Frequência de nascimentos (minutos): '
       ))
    morre = eval(input('Frequência de falecimentos (minutos):
       '))
    emigra = eval(input('Frequência de emigração (minutos): ')
       )

    total_minutos = 365 * 24 * 60

    nascimentos_ano = total_minutos // nasce
    mortes_ano = total_minutos // morre
    emigrantes_ano = total_minutos // emigra

    final_ano = pop + nascimentos_ano - mortes_ano -
       emigrantes_ano
    print('Resumo dos dados:')
```

```
    print('-----------------')
    print('Frequência de nascimentos: %d\nFrequência de mortes
       : %d\n\
    Frequência de emigrantes:%d\nPopulação Inicial:%d'%(nasce,
       morre, emigra, pop))
    print('Estimativa:')
    print('-----------')
    print('A população ao fim de um ano: %d' %final_ano)
```

Capítulo 5

Instruções de controlo

Exercício 5.1 F

Solução

```
def ordena_3(x,y,z):
  if ( x <= y) and (x <= z):
    if y <= z:
      return (x,y,z)
    else.
      return (x,z,y)
  if (y <= x) and (y <= z):
    if x <= z:
      return (y,x,z)
    else:
      return (y,z,x)
  if (z <= x) and (z <= y):
    if x <= y:
      return (z,x,y)
    else:
      return (z,y,x)
```

Exercício 5.2 F

Solução

```
def custo_percurso(quil, alternativa):
    if alternativa == 'A1':
        return 0.15 * quil + 6.52
    elif alternativa == 'A20':
```

```
        return 0.12 * quil + 15.2
    elif alternativa == 'A21':
        return 0.1 * quil + 5.75
```

Exercício 5.3 F

Solução

```
def vencimento(bruto, ss, cga,irs):
    """Calcula o vencimento ilíquido."""
    descontos = (ss+cdg+irs) * bruto
    return bruto - descontos
```

Exercício 5.4 F

Solução

```
def nota(e,t1,t2,t3,t4):
    """Calcula nota final."""
    nota = 0.075 * (t1 + t2 + t3 + t4) + 0.7 * e
    print(nota)
    if nota >= 14 :
        return "Aprovado"
    elif nota < 7:
        return "Reprovado"
    else:
        return "Oral"
```

Exercício 5.5 F

Solução

```
def ciclo_alternativo(n):
    for i in range(20,-1,-2):
        print("i= ",i)
```

Exercício 5.6 M

Solução

```
"""
i      j    i/j
_______________
```

```
0      1    0.0
0      2    0.0
---------------
1      1    1.0
1      2    0.5
---------------
2      1    2.0
2      2    1.0
"""
```

Exercício 5.7 M

Solução

```
def amigas(cad_1, cad_2):
    """palavras que diferem em menos de 10% dizem-se amigas.
       Assume igual comprimento"""
    # Calcula distancia
    diferem = 0
    for i in range(len(cad_1)):
        if cad_1[i] != cad_2[i]:
            diferem += 1
    percentagem = diferem / len(cad_1)
    return percentagem < 0.1
```

Exercício 5.8 M

Solução

```
def min_div(num):
    """Calcula o menor divisor de um número inteiro > 1."""
    for i in range(2,num//2 + 1):
        if num % i == 0:
            return i
    return 1

def primo(num):
    return min_div(num) == 1
```

Exercício 5.9 Módulo random M

Solução

```
import random

def dados(tentativas):
    """Determina a percentagem de lançamentos que deram um
       número par."""
    conta = 0
    for i in range(tentativas):
        numero = random.randint(0,6)
        if numero % 2 == 0:
            conta = conta + 1
    return conta/tentativas
```

Exercício 5.10 M

Solução

```
import random

def probab(num_dardos):
    """Probabilidade de acertar nas áreas ímpares."""
    conta = 0
    for i in range(num_dardos):
        x = random.uniform(0,2)
        y = random.uniform(0,2)
        if ((x <= 1) and (y >= 1)) or ((x > 1) and (y <= 1))
           or ((x > 1) and (y >= 1) and (y < x)):
            conta += 1
    return 100*conta/num_dardos
```

Exercício 5.11 F

Solução

```
def fact(n):
    """Calcula o factorial de n."""
    res = 1
    for i in range(1,n+1):
        res = res * i
    return res
```

Exercício 5.12 M

Solução

```
def seno(x,prec):
    """ Calcula o seno de x com uma dada precisão."""
    ordem = 0
    res = 0
    dif = x
    while dif> prec:
        termo= (pow(-1,ordem) * pow(x,2*ordem +1)) / fact(2*
           ordem+1)
        res,ant = res + termo,res
        dif = abs(res-ant)
        ordem = ordem + 1
    return res
```

Exercício 5.13 Módulo matplotlib **F**

Solução

```
def harmonico(n):
    """ Calcula H_n."""
    h_n = 0
    for k in range(1,n+1):
        h_n += (1/k)
    return h_n

def harmonia(n):
    """Calcula uma sequência de números harmónicos."""
    serie = ()
    for i in range(1,n+1):
        serie += (harmonico(i),)
    return serie

def main(num):
    valores = harmonia(num)
    plt.xlabel('n')
    plt.ylabel('$H_n$')
    plt.title('Números Harmónicos')
    plt.plot(valores, label='Fórmula Usual')
    plt.legend(loc=0)
    plt.show()
```

Exercício 5.14 F

Solução

```
import math
import matplotlib.pyplot as plt

def harmonico(n):
    """ Calcula H_n."""
    h_n = 0
    for k in range(1,n+1):
        h_n += (1/k)
    return h_n

def harmonico_b(n):
    return math.log(n) + 0.5772156649

def harmonia(n):
    """Calcula uma sequência de números harmónicos."""
    serie = ()
    for i in range(1,n+1):
        serie += (harmonico(i),)
    return serie

def harmonia_b(n):
    """Calcula uma sequência de números harmónicos usando
       fórmula aproximada."""
    serie = ()
    for i in range(1,n+1):
        serie += (harmonico_b(i),)
    return serie

def main(num):
    valores = harmonia(num)
    valores_b = harmonia_b(num)
    plt.xlabel('n')
    plt.ylabel('$H_n$')
    plt.title('Números Harmónicos')
    plt.plot(valores, label='Fórmula Usual')
    plt.plot(valores_b, label='Fórmula Aproximada')
```

```
    plt.legend(loc=0)
    plt.show()
```

Exercício 5.15 F

Solução

```
def exponencial(prec):
    """ Calcula o valor de 'e' com uma dada precisão. Assume
       precisão inferior a 1."""
    ordem = 0
    res = 0
    dif = 1
    while dif> prec:
        termo= 1 / fact(ordem)
        res,ant = res + termo,res
        dif = abs(res-ant)
        ordem = ordem + 1
    return res

# Variante
def exponencial_b(prec):
    """
    Calcula o valor de 'e' com uma dada precisão. Assume
       precisão inferior a 1.
    Calcula o factorial no interior do programa.
    """
    ordem = 0
    res = 0
    dif = 1
    fact = 1
    while dif> prec:
        termo= 1 / fact
        res,ant = res + termo,res
        dif = abs(res-ant)
        ordem = ordem + 1
        fact = fact * ordem
    return res
```

Exercício 5.16 M

Solução

```
def perfeito(n):
    return n == soma_div(n)

def soma_div(n):
    """Calcula a soma dos dividores de um número excluindo ele
        próprio."""
    soma = 0
    for i in range(2,n):
        if n % i == 0:
            soma += i
    return soma + 1
```

Exercício 5.17 M

Solução

```
def padrao_1(n):
    """ Imprime linhas de números entre 1 e n. Crescente
       alinhado à esquerda."""
    comp = len(str(n))
    for i in range(1,n+1):
        for j in range(1,i+1):
            print(j,end=comp*' ')
        print()

def padrao_2(n):
    """ Imprime linhas de números entre 1 e n. Decrescente
       alinhado à esquerda.."""
    for i in range(n,0,-1):
        for j in range(1,i+1):
            print(j,end=' ')
        print()

def padrao_3(n):
    """ Imprime linhas de números entre 1 e n. Crescente
       alinhado à direita."""
    for i in range(1,n+1):
        print(end=(n-i)*'  ')
        for j in range(i,0,-1):
            print(j,end=' ')
        print()
```

```
"""O primeiro padrão não tem problemas com os números."""
```

Exercício 5.18 Módulo turtle M

Solução

```
import turtle

def grelha(dim,lado):
    """Desenha uma grelha dim x dim em que cada célula tem de
       lado lado."""
    turtle.color("gray")
    tam = (dim*lado)
    x = -tam//2
    y = tam//2
    turtle.penup()
    turtle.goto(x,y)
    for lin in range(dim):
        # Desenha linha de quadrados
        for col in range(dim):
            turtle.pendown()
            quadrado(lado)
            turtle.penup()
            turtle.setx(turtle.xcor() + lado)
        # reposiciona
        turtle.penup()
        turtle.setposition(x, turtle.ycor()-lado)
    turtle.hideturtle()

def quadrado(lado):
    for i in range(4):
        turtle.fd(lado)
        turtle.rt(90)
```

Exercício 5.19 Módulo turtle M

Solução

```
import turtle

def grelha(dim,lado):
```

```
    """Desenha uma grelha dim x dim em que cada célula tem de
       lado lado."""
    turtle.color("gray")
    tam = (dim*lado)
    x = -tam//2
    y = tam//2
    turtle.penup()
    turtle.goto(x,y)
    for lin in range(dim):
        # Desenha linha de quadrados
        for col in range(dim):
            turtle.pendown()
            quadrado(lado)
            turtle.penup()
            turtle.setx(turtle.xcor() + lado)
        # reposiciona
        turtle.penup()
        turtle.setposition(x, turtle.ycor()-lado)
    turtle.hideturtle()

def quadrado(lado):
    for i in range(4):
        turtle.fd(lado)
        turtle.rt(90)

def passeio(dim, lado, passos):
    # Prepara grelha
    turtle.speed(0)
    grelha(dim,lado)
    turtle.color('red')
    turtle.home()
    turtle.pendown()
    # Passeio
    turtle.speed(6)
    turtle.dot()
    turtle.showturtle()
    lim_x = lim_y = (dim*lado)//2
    cor_x = 0
    cor_y = 0
    for i in range(passos):
        vai_para = random.choice(['N','E','S','W'])
```

```
        if (vai_para == 'N') and (cor_y < lim_y):
            cor_y += lado
            turtle.setheading(90)
            turtle.fd(lado)
        elif (vai_para == 'E') and (cor_x < lim_x):
            cor_x += lado
            turtle.setheading(0)
            turtle.fd(lado)
        elif (vai_para == 'S') and (cor_y > -lim_y):
            cor_y -= lado
            turtle.setheading(270)
            turtle.fd(lado)
        elif (vai_para == 'W') and (cor_x > -lim_x):
            cor_x -= lado
            turtle.setheading(180)
            turtle.fd(lado)
        else:
            print((vai_para,turtle.xcor(),turtle.ycor()))
            continue
```

Exercício 5.20 D

Solução

```
def is_fib(n):
    """Determina se n é um número da sequência de fibonacci.
       """
    fib_ant = 0
    fib = 1
    while fib < n:
        # next fib
        fib_ant, fib = fib, fib_ant + fib
    return fib == n
```

Exercício 5.21 F

Solução

```
def novo_index(cadeia,elemento):
    try:
        indice = cadeia.index(elemento)
```

```
        return indice
    except ValueError:
        return -1
```

Capítulo 6

Objectos (II)

Exercício 6.1 F

Solução

```
def num_id(lista_idades):
    return len(lista_idades)

def idades(lista_idades):
    return lista_idades

def idades_inv(lista_idades):
    return lista_idades[::-1]

def min_max(lista_idades):
    return min(lista_idades), max(lista_idades)

def soma_idades(lista_idades):
    return sum(lista_idades)

def abaixo_de(referencia, lista_idades):
    for elem in lista_idades:
        if elem < referencia:
            print(elem)

def tem_17(lista_idades):
    return 17 in lista_idades
```

Exercício 6.2 F

Solução

```
def pares_impares(lista):
    """ Devolve a soma dsos pares e a soma dos impares."""
    pares = 0
    impares = 0
    for elem in lista:
        if elem % 2 == 0:
            pares = pares + elem
        else:
            impares = impares + elem
    return pares, impares
```

Exercício 6.3 F

Solução

```
def alterna(lista_1, lista_2):
    """ Nova lista com elementos alternados. Assume mesmo
       comprimento."""
    nova_lista = []
    for i in range(len(lista_1)):
        nova_lista = nova_lista + [lista_1[i]] + [lista_2[i]]
    return nova_lista
```

Exercício 6.4 F

Solução

```
def conta_menores(referencia, lista):
    """ Conta o número de elementos da lista menores do que o
       de referência."""
    conta = 0
    for elem in lista:
        if elem < referencia:
            conta = conta + 1
    return conta
```

Exercício 6.5 F Módulo random

Solução

```
import random

def lanca_dados(numero):
  """
  Lança dois dados um número de vezes. Guarda resultados e
     determina
  a percentagem de somas pares.
  """
  resultados = list()
  conta = 0
  for i in range(numero):
    primo = random.randint(1,6)
    segundo = random.randint(1,6)
    resultados.append([primo,segundo])
    if ((primo + segundo) % 2) == 0:
      conta = conta + 1
  return conta/numero, resultados
```

Exercício 6.6 M

Solução

```
# Versão básica
def soma_cumulativa(lista):
  lista_1 = []
  for i in range(len(lista)):
    soma = 0
    for y in range(i+1):
      soma += lista[y]
    lista_1.append(soma)
  return lista_1

# Versão Pitónica
def soma_cumulativa_b(lista):
  lista_aux = list()
  for i in range(len(lista)):
    lista_aux.append(sum(lista[:i+1]))
  return lista_aux

# Versão ainda mais Pitónica...
```

```
def soma_cumulativa_c(lista):
    res = [sum(lista[:i+1]) for i in range(len(lista))]
    return res
```

Exercício 6.7 M

Solução

```
""" Atenção à mutabilidade!!! Usar deepcopy pois cópia de
   superfície não serve!!!."""
import copy

# Versão básica
def negativo(imagem):
    copia = copy.deepcopy(imagem)
    for linha in range(len(imagem)):
        for coluna in range(len(imagem[0])):
            if copia[linha][coluna] == 0:
                copia[linha][coluna] = 1
            else:
                copia[linha][coluna] = 0
    return copia

# Versão Pitónica
def negativo_b(imagem):
    copia = copy.deepcopy(imagem)
    for linha in range(len(imagem)):
        for coluna in range(len(imagem[0])):
            #print('antes: ',copia[linha][coluna], end=' ')
            copia[linha][coluna] = (copia[linha][coluna] + 1)
               %  2
            #print('depois: ',copia[linha][coluna])
    return copia

# Versão ainda mais Pitónica
def negativo_c(imagem):
    copia = copy.deepcopy(imagem)
    for linha in range(len(imagem)):
        for coluna in range(len(imagem[0])):
            copia[linha][coluna] ^= 1
    return copia
```

Exercício 6.8 MD

Solução

```
# Versão Básica
def roda_90(imagem):
    """Baseia-se na construção da transposta da imagem vista
       como uma matriz."""
    copia_imagem = copy.deepcopy(imagem)
    imagem_aux = list()
    # transpõe
    for coluna in range(len(copia_imagem[0])):
        nova_linha = list()
        for linha in copia_imagem:
            nova_linha += [linha[coluna]]
        imagem_aux += [nova_linha]
    # inverte dentro das linhas
    for linha in range(len(imagem_aux)):
        imagem_aux[linha] = imagem_aux[linha][::-1]

    return imagem_aux

# Versão super pitónica
def roda_90_b(imagem):
    copia = copy.deepcopy(imagem)
    transposta = zip(*copia)
    final = [list(linha[::-1]) for linha in transposta]
    return final
```

Exercício 6.9 M

Solução

```
import turtle
import random

def navega(comandos, tartaruga):
    """Simula o caminhar de uma tartaruga numa cidade
       geométrica."""
    tartaruga.color('green')
    tartaruga.dot(10)
```

```
    tartaruga.color('black')
    for comando in comandos:
        if comando == 'A':
            tartaruga.fd(30)
        elif comando == 'R':
            tartaruga.bk(30)
        elif comando =='E':
            tartaruga.left(90)
        elif comando == 'D':
            tartaruga.right(90)
        else:
            print('comando desconhecido. Foi Ignorado!')
    tartaruga.color('red')
    tartaruga.dot(10)
    tartaruga.ht()

def gera_comandos(n):
    """Gera n comandos aleatoriamente. Alguns movimentos são
       mais prováveis do que outros."""
    comandos = ''
    for i in range(n):
        if random.choice([0,0,0,1]) == 0:
            comandos += random.choice(['A','A', 'A','A','R'])
        else:
            comandos += random.choice(['E', 'D'])
    return comandos

def main_tarta(n):
    tartaruga = turtle.Turtle()
    comandos = gera_comandos(n)
    navega(comandos,tartaruga)
    turtle.exitonclick()
```

Exercício 6.10 F

Solução

```
# Criar
autor = {"php":"Rasmus Lerdorf","perl":"Larry Wall","tcl":"
   John Ousterhout","awk":"Brian Kernighan","java":"James
   Gosling","parrot":"Simon Cozens","python":"GuidovanRossum",
```

```
   "xpto":"zxcv"}
print(autor)
{'tcl': 'John Ousterhout', 'awk': 'Brian Kernighan', 'parrot':
    'Simon Cozens', 'python': 'GuidovanRossum', 'java': 'James
    Gosling', 'php': 'Rasmus Lerdorf', 'xpto': 'zxcv', 'perl':
    'Larry Wall'}
#  a) Acrescentar
autor["c++"]="stroustrup"
print(autor)
{'tcl': 'John Ousterhout', 'awk': 'Brian Kernighan', 'c++': '
   stroustrup', 'parrot': 'Simon Cozens', 'python': '
   GuidovanRossum', 'java': 'James Gosling', 'php': 'Rasmus
   Lerdorf', 'xpto': 'zxcv', 'perl': 'Larry Wall'}
# b) Alterar
autor["python"]="Guido van Rossum"
print(autor)
{'tcl': 'John Ousterhout', 'awk': 'Brian Kernighan', 'c++': '
   stroustrup', 'parrot': 'Simon Cozens', 'python': 'Guido van
    Rossum', 'java': 'James Gosling', 'php': 'Rasmus Lerdorf',
    'xpto': 'zxcv', 'perl': 'Larry Wall'}
# c) Remover
del autor["xpto"]
print(autor)
# d) Contar
len(autor)
8
{'tcl': 'John Ousterhout', 'awk': 'Brian Kernighan', 'c++': '
   stroustrup', 'parrot': 'Simon Cozens', 'python': 'Guido van
    Rossum', 'java': 'James Gosling', 'php': 'Rasmus Lerdorf',
    'perl': 'Larry Wall'}
# e) Consultar
print(autor["c++"])
'stroustrup'
```

Exercício 6.11 F

Solução

```
# Versão básica
def dicio_fruta(fruta, peso):
    """Constrói um dicionário a partir de duas listas. São
```

```
        supostas ter o mesmo comprimento e sem repetições na
        fruta."""
    base_dados = dict()
    for i in range(len(fruta)):
        base_dados[fruta[i]] = peso[i]
    return base_dados

# Versão pitónica
def dicio_fruta_b(fruta, peso):
    """Constrói um dicionário a partir de duas listas. São
        supostas ter o mesmo comprimento e em repetições na
        fruta."""
    return dict(zip(fruta,peso))
```

Exercício 6.12 M

Solução

```
def lucro(dicio):
    """ Dicionário organizado como pares (fruta, {compra: c,
        venda:v, peso:p,stock:s})."""
    lucro = 0
    for valor in dicio.values():
        lucro += (valor['peso'] - valor['stock']) * (valor['
            venda'] - valor['compra'])
    return lucro

def mais_cara(dicio):
    """Qual a fruta mais cara?"""
    lista = list()
    for fruta,dados in dicio.items():
        lista.append([dados['venda'],fruta])
    fruta_cara = max(lista)
    return fruta_cara[::-1]
```

Exercício 6.13 M

Solução

```
def data(data,dicio_s,dicio_m):
    """
```

```
    Dada uma data no formato DS/DM/M/A, e um dicionário para
       dias da semana e outro para os meses
    converte a data um formato mais compreensível.
    """
    lista = data.split("/")
    return dicio_s[int(lista[0])]+", "+lista[1]+" de "+dicio_m
       [int(lista[2])]+" de "+lista[3]
```

Exercício 6.14 M

Solução

```
import copy

def metabolismo(dicio):
    novo_dicio = copy.deepcopy(dicio)
    for chave,valor in dicio.items():
        if 'Masculino' in valor:
            metabola = 66 + 6.3 * valor[3] + 12.9 * valor[2] -
                6.8 * valor[1]
        else:
            metabola = 65.5 + 4.3 * valor[3] + 4.7 * valor[2]
               - 4.7 * valor[1]
        novo_dicio[chave].append(metabola)
    return novo_dicio
```

Exercício 6.15 M

Solução

```
def imc(dicio):
    for chave,valor in dicio.items():
        imc= valor[1] / (valor[0] ** 2)
        dicio[chave].append(imc)
    return dicio
```

Exercício 6.16 M

Solução

```
def posicoes_vogais(texto):
    """Constrói um dicionário em que as chaves são vogais e os
        valores são listas das posições onde ocorrem."""
```

```
    dicio = dict()
    for i,car in enumerate(texto):
        if car in 'aeiouAEIOU':
            dicio[car] = dicio.get(car,[]) + [i]
    return dicio
```

Exercício 6.17 D

Solução

```
# Versão Básica
def inverte(dicio):
    """ Inverte um dicionário. Admite chaves diferentes com o
       mesmo valor."""
    novo_dicio = dict()
    for chave, valor in dicio.items():
        novo_dicio[valor] = novo_dicio.get(valor, []) + [chave
           ]
    return novo_dicio

# Versão Pitónica
def inverte_dicio(dicio):
    """ Inverte dicionário no pressuposto de que chaves
       diferentes têm valores diferentes."""
    return dict([(valor, chave) for chave,valor in dicio.items
       ()])
```

Exercício 6.18 F

Solução

```
""" Árvores genealógicas organizadas como dicionário com pares
    (pai: [filho1, filho2,...])."""
def irmaos(dic,nome1,nome2):
    """ Têm o mesmo progenitor?"""
    prog1 = progenitor(dic, nome1)
    prog2 = progenitor (dic,nome2)
    return prog1 == prog2
```

Exercício 6.19 M

Solução

```
# Versão básica
def netos(dicio,progenitor):
    """ Lista netos. Filhos dos filhos"""
    desc1 = filhos(dicio,progenitor)
    if desc1:
        net = []
        for elem in desc1:
            desc2 = filhos(dicio,elem)
            if desc2:
                net = net + desc2
    else:
        return []
    return net

def filhos(dicio,progenitor):
    """ lista dos filhos."""
    res= dicio.get(progenitor,None)
    return res

# Versão funcional
def netos_b(dicio ,progenitor):
    """ Lista dos netos. Os filhos dos filhos"""
    netos = filhos_b(dicio,filhos_b(dicio,[progenitor]))
    return netos

def filhos_b(dicio,lista_progenitores):
    lista_filhos = []
    for filho in lista_progenitores:
        lista_filhos.extend(dicio.get(filho,[]))
    return lista_filhos
```

Exercício 6.20 M

Solução

```
def avo(dic,nome):
    """ Quem é o avô/avó do nome."""
    prog = progenitor(dic,nome)
    if prog:
        return progenitor(dic,prog)
    return None
```

Exercício 6.21 F

Solução

```
def conj_iguais(conj_1, conj_2):
    if len(conj_1) != len(conj_2):
  return False
    for elem in conj_1:
  if elem not in conj_2:
      return False
    return True
```

Exercício 6.22 M

Solução

```
def filtra(conj,predicado):
    lista = [elem for elem in conj if predicado(elem)]
    return set(lista)

def pred(x):
    return (x % 2 == 0)
```

Exercício 6.23 M

Solução

```
def reflexiva(relacao):
    conj = set()
    for x1,x2 in relacao:
  conj.add(x1)
  conj.add(x2)
    for elem in conj:
  if (elem,elem) not in relacao:
      return False
    return True
```

Exercício 6.24 M

Solução

```
def simetrica(relacao):
```

```
  for (x1,x2) in relacao:
if (x2,x1) not in relacao:
    return False
  return True
```

Capítulo 7

Ficheiros

Exercício 7.1 F

Solução

```
def cria_ficheiro71(nome_fich, mensagem):
    """
    Cria um ficheiro com um texto. Versão simples.
    """
    conteudo = open(nome_fich,'w')
    conteudo.write(mensagem)
    conteudo.close()

def cria_ficheiro71b(nome_fich, mensagem):
    """
    Cria um ficheiro com um texto. Versão genérica.
    """
    conteudo = open(nome_fich,'w',encoding='utf-8')
    conteudo.write(mensagem)
    conteudo.close()

frase = "Acabei de criar o meu primeiro ficheiro em Python.\n"
```

Exercício 7.2 F

Solução

```
def ler_seleccao(nome_fich, pos_inicial, num_caract):
    """
    Ler num_caract caracteres  de um ficheiro a partir
```

```
      pos_inicial. Ficheiros ASCII.
   """
   conteudo = open(nome_fich,'r')
   conteudo.seek(pos_inicial)
   cont = conteudo.read(num_caract)
   conteudo.close()
   return cont


def ler_seleccao_b(nome_fich, pos_inicial, num_caract):
   """
   Ler num_caract caracteres  de um ficheiro a partir
      pos_inicial. Geral.
   """
   conteudo = open(nome_fich,'r', encoding='utf-8')
   conteudo.read(pos_inicial)
   cont = conteudo.read(num_caract)
   conteudo.close()
   return cont
```

Exercício 7.3 F

Solução

```
def nova_linha(nome_fich, texto):
   """
   Adiciona uma linha no final ao ficheiro
   """
   conteudo = open(nome_fich,'r+')
   conteudo.seek(0,2)
   conteudo.write(texto)
   conteudo.close()

def nova_linha_b(nome_fich, texto):
   """
   Adiciona uma linha no final ao ficheiro
   """
   conteudo = open(nome_fich,'a')
   conteudo.write(texto)
   conteudo.close()
```

```
def nova_linha_c(nome_fich, texto):
    """
    Adiciona uma linha no final ao ficheiro. Geral
    """
    conteudo = open(nome_fich,'a', encoding='utf-8')
    conteudo.write(texto)
    conteudo.close()
```

Exercício 7.4 M

Solução

```
# Simples: só números separados são reconhecidos
def identifica_numeros(nome_fich):
    """
    Identifica se um ficheiro tem números, e retorna o
       resultado numa lista
    """
    f_in = open(nome_fich,'r')
    resultado =[]
    texto = f_in.read()
    texto_pal = texto.split()
    for pal in texto_pal:
       if pal.isdigit():
          resultado.append(int(pal))
    f_in.close()
    return resultado

# Básico
def identifica_numeros_b(nome_fich):
    """
    Identifica se um ficheiro tem digitos, e retorna o
       resultado numa lista
    """
    resultado =[]
    linhas = []
    conteudo = open(nome_fich,'r')
    linhas = conteudo.readlines()
    for i in range(len(linhas)):
       uma_linha = linhas[i].strip().split()
       for j in range(len(uma_linha)):
```

```
            if uma_linha[j].isdigit():
               resultado.append(int(uma_linha[j]))
      return resultado

# Mais geral
def identifica_numeros_c(ficheiro):
      """
      Identifica os números  num ficheiro e devolve-os numa lista
         .
      """
      lista_numeros = []
      f_in = open(ficheiro,'r')
      caracter = f_in.read(1)
      while caracter !='':
         num = ''
         while (not caracter.isdigit()) and (caracter != ''):
            caracter = f_in.read(1)
         if caracter != '':
            num = caracter
            caracter = f_in.read(1)
            while caracter.isdigit():
               num = num + caracter
               caracter = f_in.read(1)
            lista_numeros.append(int(num))
      f_in.close()
      return lista_numeros
```

Exercício 7.5 Módulo matplotlib M

Solução

```
import matplotlib.pyplot as plt

def temp_max_min(f_ent):
      """
      Lê temperaturas mensais de várias cidades, calcula valores
         máximos e mínimos.
      Mostra o resultado num gráfico.
      """
      # lê dados
      f_in = open(f_ent)
```

```
    dados = []
    cidade = f_in.readline()
    while cidade != '':
        dados.append([float(valor) for valor in cidade[:-1].
            split()])
        cidade = f_in.readline()
    # calcula máximo e mínimo
    lista_valores_mes = list(zip(*dados))
    maximos = [max(mes) for mes in lista_valores_mes]
    minimos = [min(mes) for mes in lista_valores_mes]
    # mostra resultados
    plt.figure(1)
    plt.plot(maximos)
    plt.plot(minimos)
    plt.show()

# Gráfico mais sofisticado...
def temp_max_min_b(f_ent):
    """
    Lê temperaturas mensais de várias cidades, calcula valores
        máximos e mínimos.
    Mostra o resultado num gráfico.
    """
    # lê dados
    f_in = open(f_ent)
    dados = []
    cidade = f_in.readline()
    while cidade != '':
        dados.append([float(valor) for valor in cidade[:-1].
            split()])
        cidade = f_in.readline()
    # calcula máximo e mínimo
    lista_valores_mes = list(zip(*dados))
    maximos = [max(mes) for mes in lista_valores_mes]
    minimos = [min(mes) for mes in lista_valores_mes]
    # mostra resultados
    plt.figure(1)
    absissa = range(len(maximos))
    plt.xlabel('Meses')
    plt.ylabel('Temperaturas')
    plt.title('Max & Min')
```

```
    plt.plot(absissa,maximos,'r-o', label='Máximos')
    plt.plot,(absissa, minimos,'b-^', label='Mínimos')
    plt.legend(loc='best')
    plt.show()

def temp_max_min_c(ficheiro):
    """
    Lê temperaturas mensais de várias cidades, calcula valores
       máximos e mínimos.
    Mostra o resultado num gráfico. Recorre a listas por
       compreensão.
    """
     # ler dados
     f_in = open(ficheiro,'r')
     dados = f_in.readlines()
     dados = [[float(temp) for temp in linha[:-1].split()] for
        linha in dados]
     dados = list(zip(*dados))
     # calcula máximos e mínimos
     maximos = [max(mes) for mes in dados]
     minimos = [min(mes) for mes in dados]
     # visualizar
     plt.title('Max & Min')
     plt.xlabel('Meses')
     plt.ylabel('Temperaturas')
     plt.plot(maximos, label='Máximos')
     plt.plot(minimos,label='Mínimos')
     plt.legend(loc='best')
     plt.show()
     return dados
```

Exercício 7.6 M

Solução

```
def copia_ficheiro(fich_1,fich_2):
    """ Copia do primeiro para o segundo."""
    fich1 = open(fich_1, "r",encoding='utf-8')
    fich2 = open(fich_2, "w",encoding='utf-8')
    val = fich1.read()
    fich2.write(val)
```

```
    fich1.close()
    fich2.close()

def copia_ficheiro_bin(fich_1,fich_2):
    """ Caso de ficheiros binários."""
    fich1 = open(fich_1, "rb")
    fich2 = open(fich_2, "wb")
    val = fich1.read()
    fich2.write(val)
    fich1.close()
    fich2.close()
```

Exercício 7.7 Módulo random Módulo turtle M

Solução

```
import turtle
import random

def gera_pares(n):
   res = [(random.randint(1,6), random.randint(1,6)) for i in
      range(n)]
   return res

def cria_ficheiro(nome,dados):
   f_out = open(nome,'w')
   for par in dados:
      linha = str(par[0]) + '\t' + str(par[1]) + '\n'
      f_out.write(linha)
   f_out.close()

def le_ficheiro_77(nome):
   f_in = open(nome,'r')
   dados = []
   for linha in f_in:
      x,y = linha[:-1].split()
      dados.append([int(x), int(y)])
   f_in.close()
   return dados
```

```
def visualiza(tartaruga,dados):
   tartaruga.up()
   tartaruga.goto(dados[0])
   tartaruga.down()
   for ponto in dados[1:]:
      tartaruga.goto(ponto)
   turtle.hideturtle()

def main77(ficheiro, num_pontos):
   # Sistema de Coordenadas
   turtle.setworldcoordinates(0,0,7,7)
   # Formação dos dados
   pares = gera_pares(num_pontos)
   cria_ficheiro(ficheiro,pares)
   # Leitura dos dados
   pares_tartaruga = le_ficheiro_77(ficheiro)
   # Visualização
   tartaruga= turtle.Turtle()
   visualiza(tartaruga,pares_tartaruga)
   turtle.exitonclick()
```

Exercício 7.8 Módulo matplotlib Módulo turtle M

Solução

```
# --- comum às duas versões

def le_ficheiro_78(nome):
   """Lê e transforma os pares na soma."""
   f_in = open(nome,'r')
   dados = []
   linha =  f_in.readline()
   while linha !='' and linha != '\n':
      x,y = linha[:-1].split()
      dados.append((int(x),int(y)))
      linha = f_in.readline()
   f_in.close()
   return dados


def analisa_frequencias_78(dados):
```

```
   """Constrói dicionário de frequências."""
   frequencias={}
   for valor in dados:
      frequencias[valor] = frequencias.get(valor,0)+1
   return frequencias

# ----- versão matplotlib

import matplotlib.pyplot as plt

def visualiza_frequencias_plt_78(frequencias):
   """Recurso a matplotlib."""

   chaves = list(frequencias.keys())
   x = list(range(len(chaves)))
   etiquetas = [str(elem) for elem in chaves]
   valores = list(frequencias.values())
   plt.xticks(x,etiquetas, rotation=45)
   plt.bar(x,valores)
   plt.title('Gráfico de Ocorrências')
   plt.xlabel('Números')
   plt.ylabel('Ocorrências')
   plt.show()


def main78_plt(ficheiro):
   # Leitura dos dados
   pares = le_ficheiro_78(ficheiro)
   # Análise de frequências
   frequencias = analisa_frequencias_78(pares)
   # Visualização de frequências
   visualiza_frequencias_plt_78(frequencias)

# ----- versão  turtle

import turtle

def desenha_coluna(numero, altura):
   x=(numero*16) - 200 # para começar no inicio do eixo
   y=altura * 20
```

```
    # escrever número no eixo
    turtle.up()
    turtle.goto(x+5,-20)
    turtle.down()
    turtle.write(str(numero), move=False, align='left', font=('
        Arial', 10, 'bold'))

    #desenhar coluna
    turtle.up()
    turtle.goto(x+2,0)
    turtle.down()
    turtle.goto(x+2,y)
    turtle.goto(x+14,y)
    turtle.goto(x+14,0)

    # escrever número da frequência
    turtle.up()
    turtle.goto(x+5,y+5)
    turtle.down()
    turtle.write(str(altura), move=False, align='left', font=('
        Arial', 10, 'normal'))


def visualiza_frequencias_turtle_78(frequencias):
     # desenhar eixo dos x
    turtle.up()
    turtle.goto(-200,0)
    turtle.down()
    turtle.goto(10*len(frequencias),0)
    # desenhar frequências

    for i,ch in enumerate(frequencias):
        desenha_coluna(i,frequencias.get(ch,0))

    turtle.hideturtle()

def main78_turtle(ficheiro):
    # Leitura dos dados
    pares = le_ficheiro_78(ficheiro)
    # Análise de frequências
    frequencias = analisa_frequencias_78(pares)
```

```
   # Visualização de frequências
   visualiza_frequencias_turtle_78(frequencias)
   turtle.exitonclick()
```

Exercício 7.9 Módulo matplotlib Módulo turtle M

Solução

```
def le_ficheiro_79(nome):
   """Lê e transforma os pares na soma."""
   f_in = open(nome,'r')
   dados = []
   linha =  f_in.readline()
   while linha !='' and linha != '\n':
      x,y = linha[:-1].split()
      total = int(x)+int(y)
      dados.append(total)
      linha = f_in.readline()
   f_in.close()
   return dados


def analisa_frequencias_79(dados):
   """Constrói dicionário de frequências."""
   total_num = len(dados)
   frequencias={}
   for valor in dados:
      frequencias[valor] = frequencias.get(valor,0)+1
   for ch in frequencias:
      frequencias[ch] /= total_num
   return frequencias

# ----- versão matplotlib
import matplotlib.pyplot as plt

def visualiza_frequencias_plt(frequencias):
   """Recurso a matplotlib."""
   for ch in frequencias:
      plt.bar(ch,frequencias[ch])
```

```
    plt.xticks(list(frequencias.keys()))
    plt.title('Gráfico de Frequências')
    plt.xlabel('Números')
    plt.ylabel('Frequência')
    plt.show()



def main79_plt(ficheiro):
    # Leitura dos dados
    pares = le_ficheiro_79(ficheiro)
    # Análise de frequências
    frequencias = analisa_frequencias_79(pares)
    # Visualização de frequências
    visualiza_frequencias_plt(frequencias)

# ----- versão  turtle
import turtle

def desenha_coluna(tartaruga, numero, altura):
    x=(numero-1)*20 # para começar no inicio do eixo
    y=altura *500

    # escrever número no eixo
    tartaruga.up()
    tartaruga.goto(x+5,-20)
    tartaruga.down()
    tartaruga.write(str(numero), move=False, align='left', font
        =('Arial', 10, 'bold'))

    #desenhar coluna
    tartaruga.up()
    tartaruga.goto(x+2,0)
    tartaruga.down()
    tartaruga.goto(x+2,y)
    tartaruga.goto(x+18,y)
    tartaruga.goto(x+18,0)

    # escrever número da frequência
    tartaruga.up()
    tartaruga.goto(x+5,y+5)
```

```
    tartaruga.down()
    tartaruga.write(str(altura), move=False, align='left', font
        =('Arial', 10, 'normal'))


def visualiza_frequencias_turtle(tartaruga,frequencias):
     # desenhar eixo dos x
    tartaruga.up()
    tartaruga.goto(0,0)
    tartaruga.down()
    tartaruga.goto(20*12,0)

    # desenhar frequências
    for i in range(2,13):
        desenha_coluna(tartaruga,i,frequencias.get(i,0))
    tartaruga.hideturtle()

def main79_turtle(ficheiro):
    # Leitura dos dados
    pares = le_ficheiro_79(ficheiro)
    # Análise de frequências
    frequencias=analisa_frequencias_79(pares)
    print(frequencias)
    # Visualização de frequências
    tartaruga= turtle.Turtle()
    visualiza_frequencias_turtle(tartaruga,frequencias)
    turtle.exitonclick()

if __name__ == '__main__':
    prefixo = '/Users/ernestojfcosta/data/'
    main79_turtle(prefixo+'exo_77.txt')
```

Exercício 7.10 Módulo matplotlib Módulo turtle M

Solução

```
import matplotlib.pyplot as plt

def le_ficheiro710(nome):
    f_in = open(nome,'r', encoding='utf-8')
    dados = f_in.read()
```

```
   dados = dados.lower()
   f_in.close()
   return dados

def analisa_frequencias710(dados):
   sinais = [' ', '.', ',','!','?','\n','-','_','(',')','1','2
      ','3','4','5','6','7','8','9','0']
   especiais = {'é':'e','á':'a','ç':'c','ó':'o','ã':'a', 'í':'
      i','ê':'e','ô':'o','ò':'o','õ':'o'}
   frequencias={}
   for caractere in dados:
      if caractere not in sinais:
          if caractere not in especiais:
              frequencias[caractere]=frequencias.get(caractere
                 ,0)+1
          else:
              frequencias[especiais[caractere]] = frequencias.
                 get(especiais[caractere],0)+1
   return frequencias


def visualiza_frequencias710(ocorrencias):
   """ A partir do dicionário das ocorrências produz o plot
   das percentagens.
   """
   lista_ocorrencias = list(ocorrencias.items())
   lista_ocorrencias.sort()
   total = sum(list(ocorrencias.values()))


   nomes = [valor[0] for valor in lista_ocorrencias]
   percentagem = [100* valor[1]/total for valor in
      lista_ocorrencias]
   plt.title('Frequência de Ocorrências')
   plt.ylabel('Percentagem')
   plt.xlabel('Caracteres')
   plt.grid(True)
   plt.xticks(range(len(nomes)), nomes)
   plt.plot(percentagem)
   plt.show()
```

```
def main710(ficheiro):
   # Leitura dos dados
   dados = le_ficheiro710(ficheiro)
   # Frequências
   frequencias = analisa_frequencias710(dados)
   # Visualização de frequências
   visualiza_frequencias710(frequencias)
```

Exercício 7.11 M

Solução

```
def main711(ficheiro,transacao,vendedor):
   """ Actualiza um ficheiro de vendas."""
   # Abre ficheiro  e posiciona-se
   with open(ficheiro,'a') as f_in:
      # Actualiza
      nova_transac = '%d,%s,%d,%s,%.2f\n' % transacao
      f_in.write(nova_transac)
      f_in.close()
      # Mostra
      print('Venda a dinheiro nº %d   ' % transacao[0])
      print('*' * 30)
      dados = transacao[1:] + (vendedor,)
      print('Empresa: %s\nN.C.: %d\nData:  %s\nValor: %.2f
         Euros\nVendedor: %s' % dados)
```

Exercício 7.12 M

Solução

```
def main713(ficheiro, dicio_profs, dicio_estados):
   """
   A partir de  um ficheiro com dados pessoais codificados
      constrói um novo baseado
   num dicionário de códigos de profissões e num dicionário de
       códigos de estados civis.
   """
   with open(ficheiro, 'r',encoding='utf-8') as f_in:
```

```
    f_out = open(prefixo+'pessoas.txt','w', encoding='utf-8'
        )
    linha = f_in.readline()
    while linha !='':
        linha = linha[:-1].split(',')
        nova_linha = linha[0]+','+linha[1]+','+str(
            dicio_profs[int(linha[2])])+','+ str(dicio_estados
            [int(linha[3])])+'\n'
        f_out.write(nova_linha)
        linha = f_in.readline()
    f_out.close()
    f_in.close()
```

Exercício 7.13 D

Solução

```
def correlacao(fich_1, fich_2, numero):
    """ Calcula o coeficiente de correlação entre dados de duas
        acções."""
    dados_fich_1 = busca_dados(fich_1, numero)
    dados_fich_2 = busca_dados(fich_2, numero)
    valor = pearson(dados_fich_1, dados_fich_2)
    return valor

def busca_dados(fich,numero):
    """Retira os dados da cotação de fecho para o formato dado.
       """
    fich_ficheiro = open(fich, 'r')
    dados_ficheiro = fich_ficheiro.readlines()[1:numero]
    fich_ficheiro.close()
    # retira dados
    dados_ficheiro_fecho = []
    for linha in dados_ficheiro:
        dados_ficheiro_fecho.append(float(linha.split(',')[4]))
    return dados_ficheiro_fecho

def pearson(lista_a, lista_b):
    """ Calcula o coeficiente de correlação entre duas listas
        de valores."""
    media_a = media(lista_a)
```

```
    media_b = media(lista_b)
    desvio_a = desvio_padrao(lista_a)
    desvio_b = desvio_padrao(lista_b)
    n = len(lista_a)

    soma = 0
    for indice in range(n):
        soma = soma + (lista_a[indice] - media_a) * (lista_b[
            indice] - media_b)
    correlacao = float(soma) / ((n - 1) * desvio_a * desvio_b)

    return correlacao

def media(lista):
    """Calcula a média associada aos valores na lista."""
    return sum(lista)/float(len(lista))


def desvio_padrao(lista):
    """Calcula o desvio padrao dos elementos na lista."""
    a_media = media(lista)
    soma = 0.0
    for elem in lista:
        soma = soma + (elem - a_media) ** 2
    desvio = math.sqrt(float(soma)/ (len(lista) - 1))
    return desvio
```

Exercício 7.14 D

Solução

```
def gera_carta(carta,clientes):
    """
    carta = texto geral da carta, num ficheiro.
    clientes = dicionário com os dados dos clientes.
    nome, data de nascimento (dd/mm/aaaa), morada, telefone
    """
    PREFIXO = '/tempo/data/'
    # lê carta
    f_in = open(carta,'r')
    texto_carta = f_in.read()
```

```
    f_in.close()
    # filtra clientes
    lista_clientes = [(nome_cliente(cliente),morada_cliente(
        cliente)) for cliente in clientes.values() if (
        ano_cliente(cliente) < 1974)]


    for numero in range(len(lista_clientes)):
        # processa
        saudacao = 'Caro(a)'
        nome = lista_clientes[numero][0]
        morada = lista_clientes[numero][1]
        preambulo = saudacao+' '+nome + '\n' + morada + '\n\n'

    f_out = open(PREFIXO+nome+str(numero)+'.txt','w')
    f_out.write(preambulo)
    f_out.write(texto_carta)
    f_out.close



def dados(cliente):
    nome,data,morada,telefone = cliente
    dia, ano, mes = data.split('/')
    return (nome, (int(dia),int(ano),int(mes)),morada, int(
        telefone))

def ano_cliente(cliente):
    dados_cliente = dados(cliente)
    ano = dados_cliente[1][2]
    return ano

def nome_cliente(cliente):
    dados_cliente = dados(cliente)
    nome = dados_cliente[0]
    return nome

def morada_cliente(cliente):
    dados_cliente = dados(cliente)
    morada = dados_cliente[2]
    return morada
```

```
def main716():
   clientes = {100:('Ernesto','15/06/1953','F 26','239790019')
      ,101:('Joana','29/09/2001','A  15','239700400'),102: ('
      Lurdes','17/06/1913','G 30','808242424'),103:('Daniela',
      '31/03/2002','F 16','239400400')}
   gera_carta('/tempo/data/carta.txt', clientes)
```

Exercício 7.15 D

Solução

```
def acrescenta_musica(ficheiro, musica):
   with open(ficheiro,'a',encoding='utf-8') as f_in:
      f_in.seek(0,2)
      musica = musica + ('Não',)
      nova_musica = '\n%s\t%s\t%s\t%s\t%s' % musica
      f_in.write(nova_musica)
      f_in.close()


def empresta_musica(ficheiro,nome_musica):
   """ Anota uma música existente como emprestada."""
   with open(ficheiro,'r+',encoding='utf-8') as f_in:
      # Procura Música
      linha = f_in.readline()
      while linha != '':
         comp = len(linha)
         linha = linha[:-1].strip().split()
         if linha[1] == nome_musica:
            break
         linha = f_in.readline()
      else:
         print('Música inexistente!')
         return 'Done'
      # Actualiza
      linha[-1] = 'Sim'
      f_in.seek(f_in.tell() - comp - 1)
      nova_linha = '\t'.join(linha) + '\n'
      f_in.write(nova_linha)
      f_in.close()
```

```
def mostra_musicas_tipo(ficheiro,tipo):
      with open(ficheiro,'r+',encoding='utf-8') as f_in:
         # Procura Música
         linha = f_in.readline()
         while linha != '' and linha != '\n':
            linha = linha[:-1].strip().split()
            if linha[2] == tipo:
               print(linha)
            linha = f_in.readline()
         f_in.close()
```

Exercício 7.16 D

Solução

```
import csv

def le_csv(nome_fich):
   """
   Lê  um ficheiro em formato csv.
   """
   fich = open(nome_fich)
   csv_reader = csv.reader(fich)
   dados = []
   for linha in csv_reader:
      dados.append(linha)
   fich.close()
   return dados

def main718(nome_ficheiro):
   dados = le_csv(nome_ficheiro)
   dicio = dict()
   for elem in dados:
      classe = elem[-1]
      nome = elem[0]
      dicio[classe] = dicio.get(classe,[]) + [nome]
   return dicio
```

Exercício 7.17 Módulo csv D

Solução

```
import csv

def le_csv(nome_fich):
    """
    Lê  um ficheiro em formato csv.
    """
    fich = open(nome_fich)
    csv_reader = csv.reader(fich)
    dados = []
    for linha in csv_reader:
        dados.append(linha)
    fich.close()
    return dados

def main718(nome_ficheiro):
    dados = le_csv(nome_ficheiro)
    dicio = dict()
    for elem in dados:
        classe = elem[-1]
        nome = elem[0]
        dicio[classe] = dicio.get(classe,[]) + [nome]
    return dicio
```

Exercício 7.18 Módulo urllib D

Solução

```
import urllib.request

def extract(text, sub1, sub2):
    """ Retira o texto entre duas subcadeias."""
    return text.split(sub1, 1)[-1].split(sub2, 1)[0]

def main719(url):
    fp = urllib.request.urlopen(url)
    mybytes = fp.read()
    encoding = extract(str(mybytes).lower(), 'charset=', '"')
    if encoding:
        #mystr = mybytes.decode(encoding)
        print('-'*50)
        print( "Encoding type = %s" % encoding )
```

```
      print('-'*50)
   else:
      print("Encoding type not found!")
   fp.close()
```

Capítulo 8

Visões (II)

Exercício 8.1 F

Solução

```
def mostra_tri_inf(matriz):
    """Matriz triangular superior.Indexação pela posição."""
    for pos_linha in range(len(matriz)):
        for pos_coluna in range(0,pos_linha+1):
            print("%4d" % matriz[pos_linha][pos_coluna],end=''
               )
        print()
```

Exercício 8.2 M

Solução

```
def sub_matriz(matriz, linha, coluna, dim_x, dim_y):
    """Extrai a sub-matriz a partir de (linha, coluna) com
       dimensão dimX X dimY."""
    sub = cria_mat(dim_x, dim_y,0)
    for x in range(dim_x):
        for y in range(dim_y):
            sub[x][y] = matriz[linha+x][coluna+y]
    return sub
```

Exercício 8.3 F

Solução

```
def matriz_tuplos(altura, largura):
```

```
    """Cria uma matriz de tuplos RGB."""
    matriz = list()
    for lin in range(altura):
        linha = list()
        for col in range(largura):
            r = random.randint(0,255)
            g = random.randint(0,255)
            b = random.randint(0,255)
            linha.append((r,g,b))
        matriz.append(linha)
    return matriz
```

Exercício 8.4 M

Solução

```
def desenha_recta(janela, x1,y1,x2,y2):
    largura = janela.getWidth()
    altura = janela.getHeight()
    imagem = cImage.EmptyImage(largura,altura)

    pixel = cria_random_pixel()
    # desenha recta
    for x in range(x1,x2):
        y = int(((y2 - y1) / (x2 -x1)) * ( x - x1) + y1)
        imagem.setPixel(x,y,pixel)
    imagem.draw(janela)
```

Exercício 8.5 M

Solução

```
def arco(x,y, raio, amplitude):
    """
    Desenha (um arco de circunferência) com centro em (x,y) e
       raio.
    """
    largura = 4 * raio
    altura = 4 * raio
    janela = cImage.ImageWin('Janela', largura, altura)
    imagem = cImage.EmptyImage(largura,largura)
    pixel_branco = cImage.Pixel(255,255,255)
```

```
    for coluna in range(largura):
        for linha in range(altura):
            imagem.setPixel(coluna,linha,pixel_branco)

    #imagem.setPosition(x,y)
    pixel = cria_random_pixel()

    for angulo in range(amplitude):
        cordx = int((raio * math.cos((math.pi /float(180)) *
           angulo)) + x)
        cordy = int((raio * math.sin((math.pi /float(180)) *
           angulo))  + y)
        imagem.setPixel(cordx,cordy,pixel)
    imagem.draw(janela)
    janela.exitOnClick()
```

Exercício 8.6 F

Solução

```
def cria_moldura(imagem,tamanho, cor):
    """Cria uma moldura à volta da imagem original de tamanho
       e cor."""
    img = cImage.FileImage(imagem)
    largura = img.getWidth()
    altura = img.getHeight()

    nova_imagem = cImage.EmptyImage(largura+2*tamanho, altura
       + 2*tamanho)

    janela = cImage.ImageWin('Moldura', largura+2*tamanho,
       altura + 2*tamanho)
    janela.setBackground('red')

    img.setPosition(tamanho, tamanho)
    img.draw(janela)
    janela.exitOnClick()
```

Exercício 8.7 F

Solução

```
def corta_imagem(img, pos_x, dim_x, pos_y, dim_y):
    """ Corta uma parte da imagem a partir da posição com as
       dimensões especificadas."""
    nova_imagem = cImage.EmptyImage(dim_x, dim_y)
    for coluna in range(pos_x, pos_x+dim_x):
        for linha in range(pos_y, pos_y+dim_y):
            pixel = img.getPixel(coluna, linha)
            nova_imagem.setPixel(coluna - pos_x, linha - pos_y
               , pixel)
    return nova_imagem

def mostra_corta_imagem(imagem, pos_x, dim_x, pos_y, dim_y):
    img = cImage.FileImage(imagem)
    altura = img.getHeight()
    largura = img.getWidth()
    img_cortada = corta_imagem(img,pos_x, dim_x, pos_y, dim_y)

    janela = cImage.ImageWin('Corta Imagem', 2*largura, altura
       )
    img_cortada.setPosition(largura+pos_x, pos_y)
    img.draw(janela)
    img_cortada.draw(janela)

    janela.exitOnClick()
```

Exercício 8.8 M

Solução

```
def pixel_cinzento(pixel):
    """ Converte um pixel para escala de cinzentos tendo em
       atenção a diferença dos canais."""
    vermelho = pixel.getRed()
    verde = pixel.getGreen()
    azul = pixel.getBlue()
    int_media = int(0.299*vermelho + 0.587*verde + 0.114*azul)
        // 3
    novo_pixel = cImage.Pixel(int_media,int_media, int_media)
    return novo_pixel

def preto_branco_pixel(pixel_cinzento,limiar):
```

```
    preto = cImage.Pixel(0,0,0)
    branco = cImage.Pixel(255,255,255)
    if pixel_cinzento.getRed() < limiar :
        novo_pixel = preto
    else:
        novo_pixel = branco
    return novo_pixel

def imagem_preto_branco(imagem, limiar):
    """
    Transforma para escala de cinzentos e depois converte
    para preto e branco de acordo com o limiar.
    """
    largura = imagem.getWidth()
    altura = imagem.getHeight()
    nova_imagem=cImage.EmptyImage(largura,altura)
    for coluna in range(largura):
        for linha in range(altura):
            pixel = imagem.getPixel(coluna,linha)
            novo_pixel = preto_branco_pixel(pixel_cinzento(
               pixel),limiar)
            nova_imagem.setPixel(coluna,linha,novo_pixel)
    return nova_imagem

def mostra_preto_branco(imagem, limiar):
    """ Substitui na imagem as cores pelas mais próximas na
       palete."""
    # Cria imagens
    img = cImage.FileImage(imagem)
    nova_img = imagem_preto_branco(img, limiar)
    # Cria janela
    largura = img.getWidth()
    altura = img.getHeight()
    janela = cImage.ImageWin('Preto e Branco', 2 * largura,
       altura )
    # Coloca imagens
    nova_img.setPosition(largura+1,0)
    img.draw(janela)
    nova_img.draw(janela)
    # Termina
    janela.exitOnClick()
```

Exercício 8.9 M

Solução

```
def manipula_cor(imagem_fich, val_r,val_g,val_b):
    """ Manipula cores. Altera de modo independente os
    três canais de cor."""

    imagem = cImage.FileImage(imagem_fich)
    largura = imagem.getWidth()
    altura = imagem.getHeight()

    janela = cImage.ImageWin('Manipula cores', 2* largura,
       altura)
    imagem.draw(janela)
    nova_imagem=cImage.EmptyImage(largura,altura)

    for coluna in range(largura):
        for linha in range(altura):
            pixel= imagem.getPixel(coluna,linha)
            r = pixel.getRed()
            g = pixel.getGreen()
            b = pixel.getBlue()
            novo_r = max(r, r + ((val_r * r) % 255))
            novo_g = max(g, g + ((val_g * g) % 255))
            novo_b = max(r, r + ((val_b * b) % 255))
            novo_pixel = cImage.Pixel(novo_r,novo_g, novo_b)
            nova_imagem.setPixel(coluna,linha,novo_pixel)
    nova_imagem.setPosition(largura+1,0)
    nova_imagem.draw(janela)
    janela.exitOnClick()
```

Exercício 8.10 D

Solução

```
def distorcer_b(imagem, factor_x, factor_y):
    """ Encolhe uma imagem de acordo com os factores indicados
        ."""
    # Cria imagens
    img = cImage.FileImage(imagem)
```

```
    nova_img = encolher(img, factor_x,factor_y)
    # Cria janela
    largura = img.getWidth()
    altura = img.getHeight()
    nova_largura = largura//factor_x
    nova_altura = largura//factor_y
    janela = cImage.ImageWin('Distorce',  largura +
       nova_largura,  altura)
    # Coloca imagens
    img.draw(janela)
    nova_img.setPosition(largura + 1,0)
    nova_img.draw(janela)
    # Termina
    janela.exitOnClick()


def encolher(imagem,factor_x, factor_y):
    """
    Encolhe a imagem de acordo com os factores.
    Estes devem ser inteiros.
    """
    largura = imagem.getWidth()
    altura = imagem.getHeight()

    nova_largura = largura // factor_x
    nova_altura =  altura // factor_y

    nova_imagem = cImage.EmptyImage(nova_largura,nova_altura)

    for coluna in range(nova_largura):
        for linha in range(nova_altura):
            pixel = imagem.getPixel(coluna * factor_x, linha *
                factor_y)
            nova_imagem.setPixel(coluna,linha, pixel)
    return nova_imagem
```

Exercício 8.11 M

Solução

```
def espelho_h_s(imagem_fich):
```

```
    """Faz o espelho horizontal de uma imagem.
    Usa a parte superior."""
    imagem = cImage.FileImage(imagem_fich)
    largura = imagem.getWidth()
    altura = imagem.getHeight()
    janela = cImage.ImageWin('Espelho Vertical Superior', 2*
       largura,altura)
    imagem.draw(janela)

    nova_imagem = cImage.EmptyImage(largura,altura)
    for coluna in range(largura):
        for linha in range(altura//2):
            pixel = imagem.getPixel(coluna, linha)
            nova_imagem.setPixel(coluna,linha,pixel)
            nova_imagem.setPixel(coluna,altura - linha - 1,
               pixel)
    nova_imagem.setPosition(largura + 1, 0)
    nova_imagem.draw(janela)
    janela.exitOnClick()
```

Exercício 8.12 M

Solução

```
def mostra_reduz_cores(imagem, palete):
    """ Substitui na imagem as cores pelas mais próximas na
       palete."""
    # Cria imagens
    img = cImage.FileImage(imagem)
    nova_img = reduz_cores(img, palete)
    # Cria janela
    largura = img.getWidth()
    altura = img.getHeight()
    janela = cImage.ImageWin('Reduz Cores', 2 * largura,
       altura )
    # Coloca imagens
    nova_img.setPosition(largura+1,0)
    img.draw(janela)
    nova_img.draw(janela)
    # Termina
    janela.exitOnClick()
```

```
def reduz_cores(imagem, palete):
    largura = imagem.getWidth()
    altura = imagem.getHeight()
    nova_imagem = cImage.EmptyImage(largura, altura)
    for coluna in range(largura):
        for linha in range(altura):
            pixel = imagem.getPixel(coluna, linha)
            novo_pixel = distancia_palete(pixel, palete)
            nova_imagem.setPixel(coluna, linha, novo_pixel)
    return nova_imagem


def distancia_cores(pixel_1, pixel_2):
    """Calcula a distância entre duas cores dada por uma
       fórmula de distância elclidiana."""
    r_1 = pixel_1.getRed()
    g_1 = pixel_1.getGreen()
    b_1 = pixel_1.getBlue()

    r_2 = pixel_2.getRed()
    g_2 = pixel_2.getGreen()
    b_2 = pixel_2.getBlue()

    r = (r_1 - r_2)**2
    g = (g_1 - g_2)**2
    b = (b_1 - b_2)**2

    return math.sqrt(r+g+b)


def distancia_palete(cor, palete):
    """Qual a cor mais próxima."""
    distancia_min = distancia_cores(cImage.Pixel(0,0,0),
       cImage.Pixel(255,255,255))
    cor_min = cor
    for pal_cor in palete:
        cor_pixel = cImage.Pixel(pal_cor[0],pal_cor[1],
           pal_cor[2])
        distancia = distancia_cores(cor,cor_pixel)
```

```
        if distancia < distancia_min:
            distancia_min = distancia
            cor_min = cor_pixel
    return cor_min
```

Exercício 8.13 M

Solução

```
def colagem(lista_imagens, largura, altura):
    """ lista_imagens = [....,[imagem, (posx, poy)], ...]"""
    # cria janela principal inicialmente vazia
    janela = cImage.ImageWin('Colagem',largura,altura)
    for imagem in lista_imagens:
        img = cImage.FileImage(imagem[0])
        img.setPosition(imagem[1][0], imagem[1][1])
        img.draw(janela)
    janela.exitOnClick()
```

Exercício 8.14 M

Solução

```
def blur(imagem):
    """Suaviza uma imagem."""
    # Inicializa
    largura = imagem.getWidth()
    altura = imagem.getHeight()
    nova_imagem = cImage.EmptyImage(largura,altura)

    # Percorre a imagem e calcula
    for coluna in range(largura):
        for linha in range(altura):
            novo_pixel = media(coluna,linha,imagem)
            nova_imagem.setPixel(coluna,linha,novo_pixel)
    return nova_imagem

def media(coluna, linha, imagem):
    """Calcula o valor médio dos pixeis na vizinhança do pixel
        (coluna,linha)."""
    largura = imagem.getWidth()
    altura = imagem.getHeight()
```

```
    # Extrai pixeis
    vizinhos = []
    for c in [-1,0,1]:
        for l in [-1,0,1]:
            nova_coluna = coluna+c
            nova_linha = linha+l
            if (0 < nova_coluna < largura) and (0 < nova_linha
                < altura):
                vizinhos.append(imagem.getPixel(nova_coluna,
                   nova_linha))
    # Calcula pixel "médio" por canal
    n_viz = len(vizinhos)
    r = sum([vizinhos[i].getRed() for i in range(n_viz)])//
       n_viz
    g = sum([vizinhos[i].getGreen() for i in range(n_viz)])//
       n_viz
    b = sum([vizinhos[i].getBlue() for i in range(n_viz)])//
       n_viz

    novo_pixel = cImage.Pixel(r,g,b)
    return novo_pixel

def mostra_blur(imagem):
    """ Procede à diminiuição da pixelização da imagem."""
    # Cria imagens
    img = cImage.FileImage(imagem)
    img_ampliada = ampliar(img,3,2)
    nova_img = blur(img_ampliada)
    # Cria janela
    largura = img.getWidth()
    altura = img.getHeight()
    janela = cImage.ImageWin('Blur', 6 * largura, 2*altura )
    # Coloca imagens
    nova_img.setPosition(3*largura+1,0)
    img_ampliada.draw(janela)
    nova_img.draw(janela)
    # Termina
    janela.exitOnClick()
```

Exercício 8.15 M

Solução

```
masc_1 = [[1/9,1/9,1/9],[1/9,1/9,1/9],[1/9,1/9,1/9]]
sharpen = [[0,-1,0],[-1,5,-1],[0,1,0]]
sharpen_2 =
   [[0,0,0,0,0],[0,0,-1,0,0],[0,-1,5,-1,0],[0,0,1,0,0],
   [0,0,0,0,0]]
sharpen_3 = [[-1,-1,-1],[-1,9,-1],[-1,-1,-1]]
blur = [[1,1,1],[1,1,1],[1,1,1]]
edge_enhance = [[0,0,0],[-1,1,0],[0,0,0]]
edge_detect = [[0,1,0],[1,-4,1],[0,1,0]]
emboss = [[-2,-1,0],[-1,1,1],[0,1,2]]
sobel_v = [[-1,0,1],[-2,0,2],[-1,0,1]]
sobel_h = [[1,2,1],[0,0,0],[-1,-2,-1]]
gauss_1 = [[1/16,2/16,1/16],[2/16,4/16,2/16],[1/16,2/16,1/16]]


# Variante em relação ao texto do livro
def convolve_todos(imagem, pix_linha, pix_coluna, kernel):
    """ Calcula a convolução de um pixel."""
    kernel_coluna_base = pix_coluna - 1
    kernel_linha_base = pix_linha - 1

    soma_r = 0
    soma_g = 0
    soma_b = 0
    for linha in range(kernel_linha_base, kernel_linha_base +
       3):
        for coluna in range(kernel_coluna_base,
           kernel_coluna_base + 3):
            k_coluna_indice = coluna - kernel_coluna_base
            k_linha_indice = linha - kernel_linha_base

            pixel = imagem.getPixel(coluna, linha)
            intensidade_r = pixel.getRed()
            intensidade_g = pixel.getGreen()
            intensidade_b = pixel.getBlue()

            soma_r = int(soma_r + intensidade_r * kernel[
               k_linha_indice][k_coluna_indice])
            soma_g = int(soma_g + intensidade_g * kernel[
```

```
                k_linha_indice][k_coluna_indice])
            soma_b = int(soma_b + intensidade_b * kernel[
                k_linha_indice][k_coluna_indice])
    return soma_r,soma_g,soma_b

def convolve_geral(imagem_fich, kernel):
    imagem = cImage.FileImage(imagem_fich)
    largura = imagem.getWidth()
    altura = imagem.getHeight()
    nova_imagem = cImage.EmptyImage(largura, altura)
    for linha in range(1, altura - 1):
        for coluna in range(1, largura - 1):
            r,g,b = convolve_todos(imagem, linha, coluna,
                kernel)
            pixel_cor = cImage.Pixel(r,g,b)
            pixel_cinza = pixel_cinzento(pixel_cor)
            nova_imagem.setPixel(coluna, linha, pixel_cor)
    janela = cImage.ImageWin('Convolve',2*largura,altura)
    imagem.draw(janela)
    nova_imagem.setPosition(largura+1,0)
    nova_imagem.draw(janela)
    janela.exitOnClick()
```

Exercício 8.16 M

Solução

```
ef elimina_ruido(imagem):
    """ Elimina o ruido de uma imagem  calculando a mediana.
        """
    # Inicializa
    largura = imagem.getWidth()
    altura = imagem.getHeight()
    nova_imagem = cImage.EmptyImage(largura,altura)
    # Percorre a imagem e calcula
    for coluna in range(1,largura-1):
        for linha in range(1,altura-1):
            novo_pixel = mediana(coluna,linha,imagem)
            nova_imagem.setPixel(coluna,linha,novo_pixel)
    return nova_imagem
```

```
def mediana(coluna, linha, imagem):
    """Calcula a mediana dos pixeis na vizinhança do pixel (
       coluna,linha)."""
    r,g,b = [], [], []
    for c in [-1,0,1]:
        for l in [-1,0,1]:
            nova_coluna = coluna+c
            nova_linha = linha+l
            pixel = imagem.getPixel(nova_coluna, nova_linha)
            # Actualiza pixel por canal
            r.append(pixel.getRed())
            g.append(pixel.getGreen())
            b.append(pixel.getBlue())
    r.sort()
    med_r = r[len(r)//2]
    med_g = g[len(g)//2]
    med_b = b[len(b)//2]
    novo_pixel = cImage.Pixel(med_r,med_g,med_b)
    return novo_pixel

def mostra(imagem):
    """ Procede à diminiuição do ruído da imagem."""
    # Cria imagens
    img = cImage.FileImage(imagem)
    nova_img = elimina_ruido(img)
    # Cria janela
    largura = img.getWidth()
    altura = img.getHeight()
    janela = cImage.ImageWin('Elimina ruído', 2 * largura,
       altura )
    # Coloca imagens
    nova_img.setPosition(largura+1,0)
    img.draw(janela)
    nova_img.draw(janela)
    # Termina
    janela.exitOnClick()
```

Exercício 8.17 M

Solução

```
def combina_pixel(pixel1, pixel2,function):
    """ Combina dois pixeis de acordo com a função."""
    r1 = pixel1.getRed()
    g1 = pixel1.getGreen()
    b1 = pixel1.getBlue()

    r2 = pixel2.getRed()
    g2 = pixel2.getGreen()
    b2 = pixel2.getBlue()

    r,g,b = function(r1,r2,g1,g2,b1,b2)
    return cImage.Pixel(r,g,b)

def media(r1,g1,b1, r2,g2,b2):
    """ Devolve tuplo formado pela média."""
    r = (r1 + r2) // 2
    g = (g1 + g2) //2
    b = (b1 + b2) // 2
    return r,g,b

def maior(r1,g1,b1, r2,g2,b2):
    """ Devolve tuplo formado pela pelo maior dos dois."""
    r = max(r1 , r2)
    g = max(g1 , g2)
    b = max(b1 , b2)
    return r,g,b

def menor(r1,g1,b1, r2,g2,b2):
    """ Devolve tuplo formado pela pelo maior dos dois."""
    r = min(r1 , r2)
    g = min(g1 , g2)
    b = min(b1 , b2)
    return r,g,b


def funde(imagem1, imagem2,  funcao):

    largura_1 = imagem1.getWidth()
    altura_1 = imagem1.getHeight()
    largura_2 = imagem2.getWidth()
    altura_2 = imagem2.getHeight()
```

```
    largura = min(largura_1,largura_2)
    altura = min(altura_1,altura_2)



    nova_imagem = cImage.EmptyImage(largura,altura)

    for coluna in range(largura):
        for linha in range(altura):
            pix1= imagem1.getPixel(coluna, linha)
            pix2 = imagem2.getPixel(coluna,linha)

            novo_pixel = combina_pixel(pix1,pix2, funcao)

            nova_imagem.setPixel(coluna,linha,novo_pixel)

    return nova_imagem



def main17(imagem_fich_1, imagem_fich_2,funcao):
    imagem_1 = cImage.FileImage(imagem_fich_1)
    imagem_2 = cImage.FileImage(imagem_fich_2)

    largura_1 = imagem_1.getWidth()
    altura_1 = imagem_1.getHeight()
    largura_2 = imagem_2.getWidth()
    altura_2 = imagem_2.getHeight()
    largura = min(largura_1,largura_2)
    altura = min(altura_1,altura_2)

    janela = cImage.ImageWin('Fusão', largura, altura)

    nova_imagem = funde(imagem_1,imagem_2,funcao)

    nova_imagem.draw(janela)
    janela.exitOnClick()
```

Exercício 8.18 D

Solução

```
def roda_90(imagem):
    """ Roda uma imagem 90 graus para a direita."""
    largura = imagem.getWidth()
    altura = imagem.getHeight()

    nova_imagem = cImage.EmptyImage(altura,largura) # troca
       por causa da rotação

    for coluna in range(largura):
        for linha in range(altura):
            pixel = imagem.getPixel(coluna,linha)
            nova_imagem.setPixel(linha,coluna,pixel)
    return nova_imagem
```

Exercício 8.19 MD

Solução

```
def roda_imagem(imagem, angulo):
    """
    Roda a imagem um valor igual a angulo. Com amputa??o.
    """
    ang_rad= math.radians(angulo)
    val1 = math.cos(ang_rad)
    val2 = math.sin(ang_rad)
    mat=[[val1, -val2],[val2, val1]]

    largura = imagem.getWidth()
    altura = imagem.getHeight()


    # cria background branco
    pixel_branco = cImage.Pixel(255,255,255)

    nova_imagem = cImage.EmptyImage(largura, altura)
    for coluna in range(largura):
        for linha in range(altura):
            nova_imagem.setPixel(coluna,linha,pixel_branco)

    # roda num novo sistema de eixos
```

```
    for coluna in range((-largura/2)+1 , largura/2):
        for linha in range((-altura/2)+1 , altura/2):

            nova_coluna, nova_linha = [int(valor) for valor in
                prod_matriz_vector(mat,[coluna, linha])]

            nova_coluna = converte_x(nova_coluna, largura//2)
            nova_linha =  converte_y(nova_linha,altura//2)

            if dentro(nova_coluna,nova_linha, largura, altura)
               :
                col = converte_x(coluna,largura//2)
                lin = converte_y(linha,altura//2)
                pixel = imagem.getPixel(col, lin)
                nova_imagem.setPixel(nova_coluna,nova_linha,
                   pixel)
    return nova_imagem

def converte_x(px, referencia):
    return referencia + px


def converte_y(py, referencia):
    return referencia - py

def dentro(px1,py1,px2,py2):
    """
    Estou a considerar o canto superior esquerdo
    como tendo coordenadas (0,0).
    """
    res = (px1 > 0) and (py1 > 0) and (px1 < px2) and (py1 <
       py2)
    return res


def main19(imagem_fich, angulo):
    imagem = cImage.FileImage(imagem_fich)

    largura = imagem.getWidth()
    altura =  imagem.getHeight()
```

```
    janela = cImage.ImageWin('Roda', largura,altura)

    nova_imagem = roda_imagem(imagem, angulo)

    nova_imagem.draw(janela)

    janela.exitOnClick()
```

Exercício 8.20 MD

Solução

```
import cImage
import random

def encripta(imagem):
    """Encripta uma imagem mudando a ordem das linhas."""
    # Converte em lista de listas

    imagem_lista = imagem.toList()
    # Define permutação
    tamanho = len(imagem_lista)
    original = list(range(tamanho))
    permuta = original[:]
    random.shuffle(permuta)
    # Encripta
    nova_imagem_lista = mistura_imagem(imagem_lista, permuta)
    # Converte de volta
    nova_imagem = cImage.ListImage(nova_imagem_lista)
    return nova_imagem

def mistura_imagem(imagem, permuta):
    nova_imagem = []
    for i in range(len(imagem)):
        nova_imagem.append(imagem[permuta[i]])
    return nova_imagem

def mostra_encripta(imagem):
    img = cImage.FileImage(imagem)
    largura = img.getWidth()
    altura = img.getHeight()
```

```
nova_imagem = encripta(img)

janela = cImage.ImageWin('Encripta', 2*largura, altura)
nova_imagem.setPosition(largura+1,0)

img.draw(janela)
nova_imagem.draw(janela)

janela.exitOnClick()
```

Capítulo 9

Recursividade

Exercícios

Exercício 9.1 F

Solução

```
def mod(n,m):
  if n < m:
    return n
  else:
    return mod(n-m,m)
```

Exercício 9.2 F

Solução

```
def prod_escalar(v,w):
    if len(v) == 0:
        return 0
    else:
        return (v[0] * w[0]) + prod_escalar(v[1:],w[1:])
```

Exercício 9.3 F

Solução

```
def pot_op(x,n):
  if n==0:
    return 1
```

```
  else:
    factor=pot_op(x,n/2)
    if (n%2 == 0):
      return factor * factor
    else:
      return factor * factor * x
```

Exercício 9.4 M

Solução

```
def remove_dup(cad):
  if len(cad) == 1:
    return cad
  elif cad[0]== cad[1]:
    return cad[0] + remove_dup(cad[2:])
  else:
    return cad[0] + remove_dup(cad[1:])
```

Exercício 9.5 F

Solução

```
def incluido(conj_1,conj_2):
    """conjuntos representados como listas."""
    if conj_1 == []:
        return True
    elif conj_1[0] not in conj_2:
        return False
    else:
        return incluido(conj_1[1:], conj_2)
```

Exercício 9.6 F

Solução

```
def intersecta(conj_1, conj_2):
  """Determina a intersecção de dois conjuntos."""
  if conj_1 == []:
    return []
  elif conj_1[0] in conj_2:
    return [conj_1[0]] + intersecta(conj_1[1:], conj_2)
  else:
```

```
    return intersecta(conj_1[1:], conj_2)
```

Exercício 9.7 M

Solução

```
def horner_rec(x, poli):
    if len(poli) == 1:
        return poli[0]
    else:
        return poli[0] + x * horner_rec(x,poli[1:])
```

Exercício 9.8 D

Solução

```
def power_set(conj):
  if conj == []:
    return [[]]
  else:
    temp = power_set(conj[1:])
    return temp + [[conj[0]] + elem for elem in temp ]
```

Exercício 9.9 D

Solução

```
def ovais(n):
    if n == 1:
        return 2
    else:
        return 2* (n - 1) + ovais(n-1)
```

Exercício 9.10 M

Solução

```
def figura_inc_lado_ang_var(lado,angulo,incl,inca):
  "Desenha recursivamente com o incremento como parâmetro"
  pd()
  if lado > 0:
    forward(lado)
    right(angulo)
```

```
    figura_inc_lado_ang(lado-incl,angulo-inca,0.8*incl,0.7*
       inca)
  ht()
  return 0
```

Exercício 9.11 F

Solução

```
from turtle import *

def arvore(lado, angulo,nivel):
  if nivel:
    pd()
    fd(lado)
    rt(angulo)
    arvore(lado/2,angulo,nivel-1)
    lt(2*angulo)
    arvore(lado/2,angulo,nivel-1)
    rt(angulo)
    bk(lado)
```

Exercício 9.12 M

Solução

```
def arv_desigual(lado, angulo, nivel):
  if nivel:
    lt(angulo)
    arv_esq(lado,angulo, nivel-1)
    rt(2*angulo)
    arv_dir(lado, angulo, nivel -1)
    lt(angulo)

def arv_esq(lado, angulo,nivel):
  fd(2*lado)
  arv_desigual(lado, angulo, nivel)
  bk(2*lado)

def arv_dir(lado, angulo,nivel):
    fd(lado)
    arv_desigual(lado, angulo, nivel)
```

```
    bk(lado)
```

Exercício 9.13 M

.

Solução

```
def aplana(L):
  if L==[]:
    return L
  elif isinstance(L[0],list):
    return aplana(L[0]) + aplana(L[1:])
  else:
    return [L[0]] + aplana(L[1:])
```

Exercício 9.14 M

Solução

```
def pat_match(pad,texto):
  if len(texto) < len(pad):
    return False
  elif pad == texto[:len(pad)]:
    return True
  else:
    return pat_match(pad,texto[1:])

# - Variante: indica o índice do começo do padrão no texto
def pat_match_ind(pad,texto,indice):
  if len(texto) < len(pad):
    return False,-1
  elif pad == texto[:len(pad)]:
    return True, indice
  else:
    return pat_match_ind(pad,texto[1:], indice +1)
```

O custo computacional é elevado. Medido pelo número de comparações é igual a $\mathcal{O}(|pad| \times |texto|)$.

Exercício 9.15 M

Solução

```
def prod_vectores(LV):
```

```
  if not LV:
    return [LV]
  else:
    res=[]
    for elem in LV[0]:
      for aux in prod_vectores(LV[1:]):
        res.append([elem] + aux)
    return res
# -- Variante
def prod_vect_2(LV):
  if not LV:
    return [LV]
  else:
    return [[elem] + aux for elem in LV[0] for aux in
       prod_vect_2(LV[1:])]
```

Exercício 9.16 M

Solução

```
from random import *
from numpy import *

# Gerador de matrizes quadradas

def gera_mat(dim, val_max):
  return array([randint(1, val_max) for i in range(dim * dim)
     ]).reshape(dim,dim)

# Strassen: caso de base
def strassen_2(X2,Y2):
  # inicia matriz resultado
  Z2=zeros((2,2), dtype=int)
  # parâmetros
  p1=(X2[0,0] + X2[1,1]) * (Y2[0,0] + Y2[1,1])
  p2= (X2[1,0] + X2[1,1]) * Y2[0,0]
  p3= X2[0,0] * (Y2[0,1] - Y2[1,1])
  p4=  X2[1,1] * (Y2[1,0] - Y2[0,0])
  p5=(X2[0,0] + X2[0,1]) * Y2[1,1]
  p6=(X2[1,0] - X2[0,0]) * (Y2[0,0] + Y2[0,1])
  p7=(X2[0,1] - X2[1,1]) * (Y2[1,0] + Y2[1,1])
```

```
  # valores actualizados
  Z2[0,0]= p1 + p4 - p5 + p7
  Z2[0,1] = p3 + p5
  Z2[1,0] = p2 + p4
  Z2[1,1] = p1 + p3 + - p2 + p6

  return Z2

# Strassen: Geral

def strassen(X,Y):
  if (X.shape == (2,2)) and (Y.shape == (2,2)):
    return strassen_2(X,Y)
  else:
    n=X.shape[0]
    Z=gera_mat(n,1)
    m= n/2
    X11 = X[:m,:m]
    X12 = X[:m,m:]
    X21 = X[m:,:m]
    X22 = X[m:,m:]

    Y11 = Y[:m,:m]
    Y12 = Y[:m,m:]
    Y21 = Y[m:,:m]
    Y22 = Y[m:,m:]

    P1= strassen((X11 + X22),(Y11 + Y22))
    P2= strassen((X21 + X22),Y11)
    P3= strassen(X11,(Y12 - Y22))
    P4= strassen( X22,(Y21 - Y11))
    P5= strassen((X11 + X12),Y22)
    P6= strassen((X21 - X11),(Y11 + Y12))
    P7= strassen((X12 - X22),(Y21 + Y22))

    Z11= P1 + P4 - P5 + P7
    Z12= P3 + P5
    Z21= P2 + P4
    Z22= P1 + P3 - P2 + P6

    Z[:m,:m] = Z11
```

```
    Z[:m,m:] = Z12
    Z[m:,:m] = Z21
    Z[m:,m:] = Z22
    return Z
```

Exercício 9.17 F

Solução

```
# Detector de paridade par

transit={'P':{'0':'P','1':'I'}, 'I':{'0':'I','1':'P'}}
inicial= 'P'
final= ['P']


def automato(estado,cad):
  if cad == '':
    return (estado in final)
  else:
    estado=transit[estado][cad[0]]
    return automato(estado,cad[1:])
```

Exercício 9.18 Módulo turtle D

Solução

```
from turtle import *

def snowflake(size,level):
  for i in range(3):
    side(size,level)
    rt(120)

def side(size,level):
  if level == 0:
    fd(size)
    return True
  else:
    side(size/3, level-1)
    lt(60)
    side(size/3, level-1)
```

```
    rt(120)
    side(size/3,level-1)
    lt(60)
    side(size/3, level-1)

def main():
  reset()
  pd()
  size=eval(input("Tamanho: "))
  nivel=eval(input('Nível: '))
  snowflake(size,level)
  ht()
  exitonclick()
```

Exercício 9.19 Módulo turtle MD

Solução

```
from turtle import *


def hilbert_esq(size,level):
  if level == 0 :
    return
  else:
    lt(90)
    hilbert_dir(size,level-1)
    fd(size)
    rt(90)
    hilbert_esq(size, level-1)
    fd(size)
    hilbert_esq(size, level-1)
    rt(90)
    fd(size)
    hilbert_dir(size,level-1)
    lt(90)

def hilbert_dir(size,level):
  if level == 0 :
    return
  else:
```

```
    rt(90)
    hilbert_esq(size,level-1)
    fd(size)
    lt(90)
    hilbert_dir(size, level-1)
    fd(size)
    hilbert_dir(size, level-1)
    lt(90)
    fd(size)
    hilbert_esq(size,level-1)
    rt(90)




def main():
  tamanho = eval(input("Tamanho: "))
  nivel = eval(input('Nível: '))
  hilbert_esq(tamanho,nivel)
  turtle. exitonclick()
```

Capítulo 10

Complementos

Exercício 10.1 F

Solução

```
>>> nome = 'ernesto'
>>> def toto():
... nome='costa'
... return None
...
>>> toto()
>>> nome
'ernesto' # <-- Resultado
```

A primeira ocorrência de nome é distinta da segunda. No primeiro caso estamos perante um nome de nível global , enquanto no segundo caso o nome é local. Como não há comunicação entre os dois espaços o valor do primeiro não é alterado, logo o valor associado continua o mesmo, ou seja, 'ernesto'.

Exercício 10.2 F

Solução

```
>>> ultima_resposta = 60
>>> def ultima_maquina():
...     global ultima_resposta
...     ultima_resposta = 'Nope!'
...     return ultima_resposta
...
>>>
>>> ultima_maquina()
```

```
'Nope!' # <--  Resultado
>>> ultima_resposta
'Nope' # <-- Resultado
```

Ao tornar o nome global as alterações reflectem-se no exterior da chamada.

Exercício 10.3 M

Solução

```
>>> def f(t=0):
...       def g(t=0):
...           def h():
...               nonlocal t
...               t += 1
...           return h, lambda:t
...       h, gt = g()
...       return h,gt,lambda:t
...
>>> h,gt,ft = f()
>>> h
<function f.<locals>.g.<locals>.h at 0x101677f28>
>>> gt
<function f.<locals>.g.<locals>.<lambda> at 0x101776048>
>>> ft
<function f.<locals>.<lambda> at 0x1017760d0>
>>> ft(),g()
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
builtins.NameError: name 'g' is not defined
>>> h()
>>> ft(),gt()
(0, 1)
```

A chamada inicial de f() origina três funções, que ficam associadas a h, a gt e a ft. Chamar g() dá erro pois não está definida ao nível mais externo. Chamar h() altera o valor do t não local, isto é do t associado à função g, mas não o associado a f(). Daí que o resultado da última chamada seja o par (0,1).

Exercício 10.4 M

Solução

```
def gere_depositos_2(saldo):
    def movimento(montante):
        nonlocal saldo
        if (montante < 0) and abs(montante) > saldo:
            return 'Saldo insuficiente'
        saldo += montante
        return saldo
    return movimento
```

Exercício 10.5 D

Solução

```
def gere_contador(inicio):
    def actualiza(accao):
        nonlocal inicio
        if accao == 'conta':
            inicio += 1
            return inicio
        elif accao == 'reiniciar':
            inicio = 0
            return inicio
        else:
            return 'Accção desconhecida'
    return actualiza
```

Exercício 10.6 D

Solução

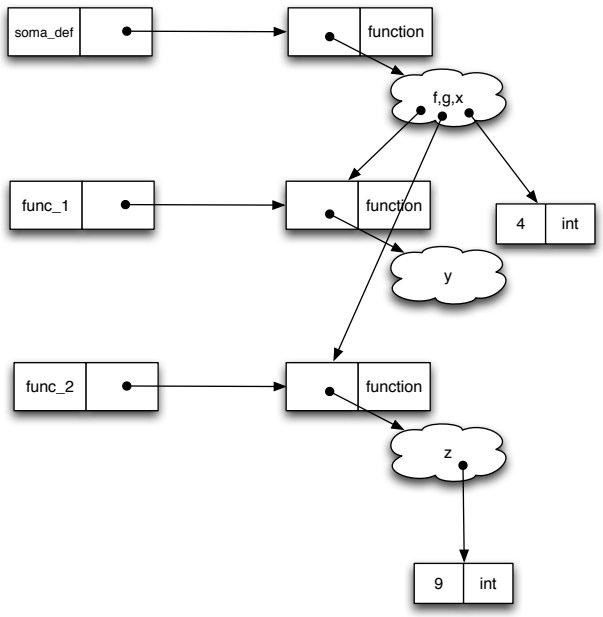
```
def eco_atrasado(frase):
    def eco():
        nonlocal velha
        while True:
            nova = get_frase()
            yield velha
            velha = nova
    velha = frase
    eco = eco()
    for i in range(3):
        print(next(eco))
    return eco
```

```
def get_frase():
    f = input('Frase: ')
    return f

def main_106():
    eco_atrasado('')
```

Exercício 10.7 M

Solução



O resultado final é igual a $64 + 9 = 73$.

Exercício 10.8 MD

Solução

```
def trace(f):
    """Como fazer o trace da execução de funções."""
    f.indent = 0
    def aux(x):
```

```
        print('|  ' * f.indent + '|__',f.__name__,x)
        f.indent += 1
        resultado = f(x)
        print('|  ' * f.indent + '|__','return', repr(
            resultado))
        f.indent -= 1
        return resultado
    return aux

def trace_b(f):
    def aux(x):
        print(f.__name__,x)
        res = f(x)
        print('return ', repr(res))
        return res
    return aux

def fib(n):
    if n < 2:
        return 1
    else:
        return fib(n-1) + fib(n-2)

def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n-1)

fib = trace(fib)
print(fib(5))
```

Exercício 10.9 F

Solução

```
def range_iter(inicio,fim,step):
    while inicio < fim:
        yield inicio
        inicio += step
```

```
for i in range_iter(3.4,8.6,0.3):
  print(i)
```

Exercício 10.10 F

Solução

```
def pares():
    num = 0
    while True:
        yield num
        num += 2
num_par = pares()
for i in range(5):
    print(next(num_par))
```

Exercício 10.11 M

Solução

```
def gera_valores(func,gera_ent):
    val = func(next(gera_ent))
    while True:
        yield val
        val = func(next(gera_ent))

def gera_ent(val,novo_val):
    while True:
        yield val
        val = novo_val(val)

def transforma(x):
    return 2*x

def funcao(x):
    return x**2


g_x = gera_ent(1,transforma)
g_val = gera_valores(funcao,g_x)
for i in range(10):
    print(next(g_val))
```

Exercício 10.12 M

Solução

```
def letras(inicial='a'):
    actual = inicial
    while 1:
        yield actual
        actual = chr(ord(actual) + 1)

gera_letras = letras('c')
for i in range(5):
    print(next(gera_letras))
```

Exercício 10.13 M

Solução

```
from math import factorial

def exponencial(x):
    val = 1
    n = 0
    while True:
        yield val
        n += 1
        val += pow(x,n)/factorial(n)

exp = exponencial(2)
for i in range(10):
    print(next(exp))
```

Exercício 10.14 M

Solução

```
def gera_par(seq):
    index = 0
    while True:
        yield (seq[index],seq[index+1])
        index += 1
```

```
def ordenada(seq):
    par = gera_par(seq)
    return all([compara(next(par)) for i in range(len(seq)-1)
        ])


def ordenada_b(seq):
    return all([compara((seq[i],seq[i+1])) for i in range(len(
        seq)-1)])

def compara(par):
    return par[0] <= par[1]
```

Exercício 10.15 M

Solução

```
def filtra(func,seq):
    return list(filter(func,seq))

def criterio(x):
    return (x % 3 != 0) and (x % 5 != 0)


lista = [1,2,3,4,5,6,7,8,9,10]
print(filtra(criterio,lista))
```

Exercício 10.16 F

Solução

```
import functools

def meu_min(sequencia):
    return functools.reduce(lambda x,y: x if x<y else y,
        sequencia)


lista = [3,6,2,8,1,-9,10]
print(meu_min(lista))
```

Exercício 10.17 F

Solução

```
import functools
import operator

def factorial(n):
    if n == 0:
        return 1
    else:
        return functools.reduce(operator.mul,range(1,n+1))

print(factorial(4))
```

Exercício 10.18 M

Solução

```
def cadeia_f(f):
    def g(x):
        def h(y):
            return f(x,y)
        return h
    return g

my_pow = cadeia_f(pow)
print(my_pow(2)(3))
```

Exercício 10.19 D

Solução

```
def cadeia_f(f):
    def g(x):
        def h(y):
            return f(x,y)
        return h
    return g
```

```
my_pow = cadeia_f(pow)
print(my_pow(2)(3))
```

Exercício 10.20 MD

Solução

```
def permutacoes(elementos):
    n = len(elementos)
    if n==0:
        yield []
    else:
        for i in range(len(elementos)):
            for perm in permutacoes(elementos[:i]+elementos[i
               +1:]):
                yield [elementos[i]] + perm


lista = [1,2,3]
pm = permutacoes(lista)
print(list(pm))
```

Capítulo 11

Desenvolvimentos

Exercício 11.1 F

Solução

```
def duplicados_3(lista):
    """
    Procura a existencia de pelo menos um par de números
       duplicados
    numa lista de inteiros positivos.
    """
    max_num = max(lista)
    tamanho = len(lista)
    aux = [0] * (max_num+1)
    print(aux)
    for i in range(tamanho):
        if aux[lista[i]] != 0:
            return True
        else:
            aux[lista[i]] = 1
    return False
```

Exercício 11.2 F

Solução

```
import random
import time
import matplotlib.pyplot as plt
```

```
def profile(f):
    """Calcula informacao sobre o tempo gasto pela computacao
       de f(x)."""
    def inner(*x):
        tempo = time.time()
        res = f(*x)
        return time.time() - tempo
    return inner

def gera_lista(tamanho, inf,sup):
    return [random.randint(inf,sup) for i in range(tamanho)]

@profile
def duplicados_1(lista):
    """
    Procura a existencia de pelo menos um par de numeros
       duplicados
    numa lista de inteiros positivos.
    """
    tamanho = len(lista)
    for i in range(tamanho-1):
        for j in range(i+1,tamanho):
            if lista[i] == lista[j]:
                return True

    return False

@profile
def duplicados_2(lista):
    """
    Procura a existencia de pelo menos um par de números
       duplicados
    numa lista de inteiros positivos.Funciona no caso de os
       números constantes
    na lista forem inferiores ao valor do comprimento da lista
       .
    """
    tamanho = len(lista)
    aux = [0] * tamanho
    for i in range(tamanho):
        if aux[lista[i]] != 0:
```

```
            return True
        else:
            aux[lista[i]] = 1
    return False


@profile
def duplicados_3(lista):
    """
    Procura a existencia de pelo menos um par de numeros
       duplicados
    numa lista de inteiros positivos.
    """
    max_num = max(lista)
    tamanho = len(lista)
    aux = [0] * (max_num+1)
    for i in range(tamanho):
        if aux[lista[i]] != 0:
            return True
        else:
            aux[lista[i]] = 1
    return False

@profile
def duplicados_4(lista):
    """Assume lista ordenada."""
    for index in range(len(lista)-1):
        if lista[index] == lista[index+1]:
            return index
    return -1

def main112():
    tempo_1 = []
    tempo_2 = []
    tempo_3 = []
    valores = [10,50,100,150, 250,
       500,750,1000,1250,2500,5000,7500,10000,12500,15000,
       20000,25000,30000,50000,100000]
    for tamanho in valores:
        max_num = int(10.0*tamanho-1)
        lista = gera_lista(tamanho,1,max_num)
```

```
        lista_2 = gera_lista(tamanho,1, tamanho-1)
        tempo_1.append(duplicados_1(lista))
        tempo_2.append(duplicados_2(lista_2))
        tempo_3.append(duplicados_3(lista))
        #lista_ord = sorted(lista[:])


    plt.title('Compara Tempos')
    plt.ylabel('Tempo Gasto')
    plt.plot(valores,tempo_1, label='Versão Normal')
    plt.plot(valores,tempo_2,label = 'Versão Lista')
    plt.plot(valores,tempo_3,label = 'Versão Lista Max')
    plt.legend(loc=2)
    plt.show()
```

Correndo o programa para diferentes valores podemos verificar que a versão que recorre a uma lista auxiliar garantindo que o maior número não é maior do que o comprimento da lista é a que apresenta melhor desempenho. Por outro lado, em geral a versão que não usa lista auxiliar é a que apresenta pior desempenho.

Exercício 11.3 M

Solução

```
import random
import time
import matplotlib.pyplot as plt

def profile(f):
    """Calcula informacao sobre o tempo gasto pela computacao
       de f(x)."""
    def inner(*x):
        tempo = time.time()
        res = f(*x)
        return time.time() - tempo
    return inner

def gera_lista(tamanho, inf,sup):
    return [random.randint(inf,sup) for i in range(tamanho)]
```

```
@profile
def duplicados_ord(lista):
    """ Pelo menos um duplicado?"""
    for index in range(len(lista)):
        if lista[index] == lista[index+1:]: # estando ordenado
            bastava fazer lista[index] == lista[index+1]
            return True
    return False

def main113():
    tempo = []
    valores = [10,50,100,150, 250,
       500,750,1000,1250,2500,5000,7500,10000,12500,15000,
       20000,25000,30000,50000,100000]
    for tamanho in valores:
        max_num = int(10.0*tamanho-1)
        lista = gera_lista(tamanho,1,max_num)
        tempo.append(duplicados_ord(lista))
    # visualiza
    plt.title('Testa Duplicados')
    plt.ylabel('Tempo Gasto')
    plt.xlabel('Dimensão')
    plt.plot(valores,tempo)
    plt.show()
```

Exercício 11.4 D

Solução

```
def ovais(n):
    """
    Em quantas regiões distintas se divide o plano com n ovais
    sabendo que as ovais se interceptam duas a duas em
       exactamente dois pontos,
    e que três ovais nunca se encontram no mesmo ponto.
    """
    if n == 1:
        return 2
    else:
        return ovais(n-1) + 2* (n-1)
```

A partir do programa podemos chegar facilmente à relação de recorrência:

$$T(n) = T(n-1) + 2 * (n-1)$$

Vamos resolver a recorrência pelo método de substituição.

$$\begin{aligned} T(n) &= T(n-1) + 2 * (n-1) \\ &= T(n-2) + 2 * (n-2) + 2 * (n-1) \\ &= T(n-k) + 2 * \sum_{i=1}^{k} (n-i) \\ &= ... \\ &= T(1) + 2 * \sum_{i=1}^{n-1} (n-i) \\ &= 2 + 2 * \sum_{i=1}^{n-1} i \\ &= 2 + 2 * \frac{(n-1) * n}{2} \\ &= 2 + n^2 - n \end{aligned}$$

Podemos então concluir que $T(n) = O(n^2)$.

Exercício 11.5 M

Solução

Não é difícil perceber que no caso pior (quando o elemento não está presente) vamos ter uma complexidade dada por:

$$T(n) = T(\frac{n}{2}) + c_1$$

Resolvendo esta recorrência por substituição (usando o facto de $n = 2^k$), obtemos :

$$T(n) = T(\frac{n}{2}) + c_1 = T(\frac{n}{2}) + c_1 T(\frac{n}{2^k}) + k * c_1 = ... = c_2 + k * c_1$$

Passando para a notação Grande O, e sabendo que $k = log_2(n)$, teremos:

$$T(n) = O(log_2(n))$$

Exercício 11.6 D

Solução

A complexidade desta abordagem pode ser medida pelas somas e multiplicações necessárias. Admitamos que para matrizes $n \times n$ temos $n = 2^k$. De acordo com as fórmulas apresentadas resulta que em relação às multiplicações temos:

$$M(k) = \begin{cases} 1 & \text{se } k = 0 \\ 7 \times M(k-1) & \text{se } k > 0 \end{cases}$$

Para as somas temos:

$$S(k) = \begin{cases} 0 & \text{se } k = 0 \\ 7 \times S(k-1) + 18 \times (2^{k-1})^2 & \text{se } k > 0 \end{cases}$$

Esta fórmula deriva do facto de para multiplicar matrizes $2^k \times 2^k$ temos que efectuar 18 somas de matrizes $2^{k-1} \times 2^{k-1}$, mais as somas envolvidas nas 7 multiplicações recursivas.

A recorrência para o caso das multiplicações é fácil de resolver:

$$\begin{aligned} M(k) &= 7 \times M(k-1) \\ &= 7^2 \times M(k-2) \\ &= ... \\ &= 7^k \times M(0) \\ &= 7^k \end{aligned}$$

Daqui resulta que $7^k = 7^{log_2 n} = n^{log_2 7} \approx n^{2.81}$. Podemos concluir por isso que $M(n) = O(n^{2.81})$. O caso das somas dá um pouco mais de trabalho, mas pode-se mostrar que $P(k) = 6 \times 7^k - 6 \times 4^k$, pelo que $S(n) \approx 6 \times n^{2.81} - 6 \times n^2$. Dadas as duas expressões (multiplicações e somas) resulta que $Strassen(n) = O(n^{2.81})$ o que compara favoravelmente com o valor usual de n^3.

Exercício 11.7 F

Solução

```
import time
import matplotlib.pyplot as plt
from math import factorial

def profile(f):
    """Calcula informacao sobre o tempo gasto pela computacao
       de f(x)."""
    def inner(*x):
        tempo = time.time()
        res = f(*x)
        return time.time() - tempo
    return inner

@profile
def exp_e(x,k):
    pot = 1
    for i in range(k):
        pot += pow(x,i+1)/factorial(i+1)
    return pot
@profile
def exp_e_2(x,k):
    pot = 1
    fact = 1
    res = 1
    for i in range(1,k):
        pot *= x
        fact *= i
        res += pot/fact
    return res


if __name__ == '__main__':
    tempo_1 = []
    tempo_2 = []
    valores = list(range(10,1000))
    for tamanho in valores:
        tempo_1.append(exp_e(123,tamanho))
        tempo_2.append(exp_e_2(123,tamanho))

    plt.title('Compara Tempos')
    plt.ylabel('Tempo Gasto')
```

```
    plt.plot(valores,tempo_1, label='Versão Func')
    plt.plot(valores,tempo_2,label = 'Versão Simples')
    plt.legend(loc=2)
    plt.show()
```

A primeira versão tem um crescimento exponencial enquanto que a segunda é linear. Este resultado era expectável devido ao recurso das funções pow e factorial.

Exercício 11.8 F

Solução

Apresentamos duas soluções para o problema. A primeira é básica e a segunda recursiva. Usando o `doctest`, ao executar o código abaixo não há a indicação de erro. Para testar a eficácia da abordagem o leitor pode introduzir exemplos errados na cadeia de comentário e verificar o resultado.

```
import doctest

def palindrome_1(objecto):
    """Determina se uma palavra é palindrome
    >>> palindrome_1('AMA')
    True
    >>> palindrome_1('TOTO')
    False
    """
    if  not isinstance(objecto,list):
        objecto = str(objecto)
    return objecto == objecto[::-1]


def palindrome_2(s):
    """Determina se uma palavra é palindrome
    >>> palindrome_1('AMA')
    True
    >>> palindrome_1('TOTO')
    False
    """
    if len(s) <= 1:
        return True
    return s[0] == s[-1] and palindrome_2(s[1:-1])

if __name__ == '__main__':
```

```
    doctest.testmod()
```

Exercício 11.9 M

Solução

Este problema tem uma solução fácil. Vamos calcular a complexidade em função do número de vezes que a instrução mais interior do código (`termo *= i`) é executada. Ela esta dentro de um ciclo que é executado i vezes. Por sua vez este ciclo é executado n vezes. Temos assim:

$$T(n) = \sum_{i=1}^{n} i = \frac{n \times (n+1)}{2}$$

Logo, podemos concluir que $T(n) = O(n^2)$.

Exercício 11.10 M

Solução

```
def max_subseq_b(seq):
    inf = 0
    sup = 0
    soma_max = 0
    for i in range(len(seq)):
        for j in range(i,len(seq)):
            soma = 0
            for k in range(i,j+1):
                soma += seq[k]
            if soma > soma_max:
                soma_max = soma
                inf = i
                sup = k
    return soma_max, inf,sup

# -- Variante pitónica
def somas_subseq(seq,i):
    res = [sum(seq[i:j+1]) for j in range(i,len(seq))]
    sup = res.index(max(res)) + i
    return max(res),i,sup

def max_subseq(seq):
```

```
    inf = 0
    sup = 0
    soma_max = 0
    for i in range(len(seq)):
        soma,n_inf, n_sup = somas_subseq(seq,i)
        if soma > soma_max:
            soma_max = soma
            inf = n_inf
            sup = n_sup
    return soma_max, inf,sup
```

Este algoritmo baseia-se num método dito de "força bruta". Considera cada ponto da sequência como possível início da subsequência pretendida e calcula a soma de todas as subsequências com início nesse ponto. No final tenta verificar se a partir desse ponto tem uma solução melhor que a melhor anterior. O invariante para o ciclo for exterior é pois fácil de definir: quando estou a analisar a partir de um dado i, em nas variáveis soma_max, inf e sup os valores correctos para todas as subsequências que começam desde o início até $i-1$. A parte mais interior faz o trabalho de manter o invariante. Quanto à complexidade. O algoritmo de força bruta tem três ciclos cuja execução é função do tamanho da sequência. Daí que se possa concluir que $T(n) = O(n^3)$. Não se pode dizer que seja famoso... Existem no entanto outras versões para o algoritmo em que usamos de algum conhecimento sobre o problema. Assim há variantes de complexidade quadrática, de $n \times \log(n)$, e mesmo um algoritmo de complexidade linear (algoritmo de Kadane). O leitor interessado pode fazer uma pesquisa por "sublista contígua de soma máxima".

Exercício 11.11 M

Solução

A correcção da primeira faz-se de modo semelhante a uma prova por indução. O caso de base corresponde a uma cadeia vazia. Por definição a inversa de uma cadeia vazia é uma cadeia vazia, pelo que a solução do caso de base está correcta. Admitindo agora que a chamada recursiva inverte correctamente a cadeia sem o último elemento, então, a juntar esse último elemento ao início da cadeia completa a inversão. No segundo exemplo a prova recorre ao conceito de invariante: depois de analisados os primeiros i caracteres da cadeia, a variável *res* contém essa cadeia invertida. O invariante é verdadeiro no início do ciclo e mantém-se verdadeiro após cada execução do ciclo. Por isso à saída a cadeia está completamente invertida. Em relação à complexidade, não é difícil de verificar que ela é linear nos dois casos. No

entanto isso não significa que demoram ambas o mesmo tempo para a mesma entrada.

Exercício 11.12 D

Solução

Vamos calcular a complexidade aproximando o número de comparações. A recorrência é semelhante à usada para as Torres de Hanói. Admitimos que $n = 2^k$.

$$T(n) = 2 \times T(\frac{n}{2}) + c_1$$

Resolvendo a recorrência obtemos:

$$\begin{aligned}
T(n) &= 2 \times T(\frac{n}{2}) + c_1 \\
&= 2 \times (2 \times T(\frac{n}{2^2}) + c_1) + c_1 = 2^2 \times T(\frac{n}{2^2}) + 2 \times c_1 + c_1 \\
&= ... \\
&= 2^k \times T(\frac{n}{2^k}) c_1 + \sum_{i=0}^{k-1} 2^i \\
&= 2^k \times T(1) + \sum_{i=0}^{k-1} 2^i \\
&= 2^k \times c_2 + 2^k - 1 \\
&= c_3 \times 2^k - 1
\end{aligned}$$

Dado que $n = 2^k$ podemos concluir que $T(n) = O(n)$.

Exercício 11.13 M

Solução

```
import random
import time
import matplotlib.pyplot as plt

def profile(f):
    """Calcula informacao sobre o tempo gasto pela computacao
       de f(x)."""
```

```
    def inner(*x):
        tempo = time.time()
        res = f(*x)
        return time.time() - tempo
    return inner


#  recursivo
def min_max_rec(lista):
    comp = len(lista)
    if comp == 1:
        return (lista[0], lista[0])
    elif comp == 2:
        return (minimo_2(lista[0], lista[1]), maximo_2(lista
           [0], lista[1]))
    else:
        meio = comp//2
        min_1, max_1 = min_max_rec(lista[:meio])
        min_2, max_2 = min_max_rec(lista[meio:])
        return (minimo_2(min_1, min_2), maximo_2(max_1, max_2)
           )

def minimo_2(x,y):
    if x < y:
        return x
    else:
        return y

def maximo_2(x,y):
    if x > y:
        return x
    else:
        return y

# porque se trata de uma definição recursiva...
@profile
def wrapper(func,*args):
    func(*args)


# iterativo
```

```
@profile
def min_max_iter(lista):
    comp = len(lista)
    if comp == 1:
        return (lista[0], lista[0])
    else:
        min_ = lista[0]
        max_ = lista[0]
        for i in range(1,comp):
            if lista[i] < min_:
                min_ = lista[i]
            if lista[i] > max_:
                max_ = lista[i]
        return (min_, max_)

# --- AUX
def gera_lista(tamanho):
    return [random.randint(1,tamanho//4) for i in range(
       tamanho)]


def main1113():
    tempo_1 = []
    tempo_2 = []

    valores = [10,50,100,150, 250,
       500,750,1000,1250,2500,5000,7500,10000,12500,15000,
       20000,25000,30000,50000,100000]
    for tamanho in valores:
        lista = gera_lista(tamanho)
        tempo_1.append(wrapper(min_max_rec,lista))
        tempo_2.append(min_max_iter(lista))



    plt.title('Compara Tempos')
    plt.ylabel('Tempo Gasto')
    plt.plot(valores,tempo_1, label='Versão Recursiva')
    plt.plot(valores,tempo_2,label = 'Versão Iterativa')
    plt.legend(loc='best')
    plt.show()
```

```
if __name__ == '__main__':
    main1113()
```

Exercício 11.14 F

Solução

O argumento de correcção é simples. O ciclo exterior pode ser caracterizado por um invariante que expressa o facto de para um dado **i** já tewr sido impressa a tabuada dos números de 1 a (i-1). No início tal é verdadeiro. Quando saímos do ciclo, do invariante e de i = (n+1) mostramos a correcção do programa. O ciclo interior também tem um invariante trivial: fixado um i, para um dado j já foram impressas as soluções de 1 até (j-1). Quando saímos deste ciclo o invariante continua verdadeiro e j = 11, pelo que fica completa a tabuada do número i. Quanto à complexidade. O ciclo interior é executado n vezes. De cada vez a instrução de impressão é executada 10 vezes. Logo a complexidade é $O(n)$.

```
def tabuada_1(n):
    """Imprime uma tabuada: (1*1) (1*2) ... (1*n)\\(2*1) (2*2)
        ... (2*n) \\ ....(n*1) ... (n*n)"""
    for i in range(1,n+1):
        # imprime tabuada do i
        pass


def tabuada_2(n):
    """Imprime uma tabuada: (1*1) (1*2) ... (1*n)\\(2*1) (2*2)
        ... (2*n) \\ ....(n*1) ... (n*n)"""
    for i in range(1,n+1):
        # imprime tabuada do i
        print('Tabuada do %d' % i)
        for j in range(1,11):
            #imprime valor (i*j)
            print('%d   X   %d  =  %d' % (i,j,i*j))
        print() # muda de linha
```

Exercício 11.15 M

Solução

Vamos começar por um esboço muito simples.

```
def insercao_1(seq):
    """Ordenamento por inserção."""
    for i in range(1,len(seq)):
        """Invariante: ordem relativa da posição 0 a (i-1)."""
        # Coloca elemento na posição i no lugar correcto entre
            0 e i.
        pass
```

Como se pode verificar o invariante é verdadeiro inicialmente. Se o código executado no interior do ciclo mantiver a verdade do invariante, à saída do ciclo, com o índice uma posição à frente da sequência, o programa faz o que é pedido. Como resolver o interior do ciclo? A ideia do ordenamento por inserção é ir testando o elemento na posição i para o colocar na sua posição correcta relativamente à parte ordenada. Daí a seguinte versão.

```
def insercao_2(seq):
    """Ordenamento por inserção."""
    for i in range(2,len(seq)):
        """Invariante: ordem relativa da posição 0 a (i-1)."""
        # Coloca elemento na posição i no lugar correcto entre
            0 e i.
        # Compara seq[i] com os que estão à sua esquerda
        elem = seq[i]
        for j in range(i-1,-1,-1):
            # procura posição correcta para seq[i] e insere
            pass
    return seq
```

O segundo ciclo for, mais interior, tem um invariante simples: para um dado i e um dado j, os valores entre as posições j+1 e i-1 são todas maiores do que o elemento na posição i. Para procurar a posição correcta efectuamos comparações e vamos deslocando os elementos uma posição para a frente enquanto forem maiores do que o elemento de comparação. Quando essa condição deixar de ser verdadeira podemos inserir o elemento na posição seguinte.

```
def insercao_3(seq):
    """Ordenamento por inserção."""
    for i in range(2,len(seq)):
        """Invariante: ordem relativa da posição 0 a (i-1)."""
        # Coloca elemento na posição i no lugar correcto entre
            0 e i.
        # Compara seq[i] com os que estão à sua esquerda
```

```
        elem = seq[i]
        for j in range(i-1,-1,-1):
            if elem < seq[j]:
                # Enquanto for menor desloca os maiores uma
                   posição para a direita
                seq[j+1] = seq[j]
            else:
                # Quando não houver nenhum insere e passa ao
                   seguinta
                seq[j+1] = elem
                break
    return seq
```

Com este código parece que resolvemos o nosso problema. Mas, por vezes, a lógica e os programas surpreendem-nos. Este programa tem um problema pois o elemento mais à esquerda pode não ser correctamente alterado. O código que resolve esta questão usa uma técnica conhecida em informática por **sentinela**: neste caso, acrescentamos à esquerda da sequência um elemento que é garantidamente mais pequeno que todos os outros.

```
def insercao_4(seq):
    """Ordenamento por inserção."""
    seq = [0] + seq[:]
    for i in range(2,len(seq)):
        """Invariante: ordem relativa da posição 0 a (i-1)."""
        # Coloca elemento na posição i no lugar correcto entre
            0 e i.
        # Compara seq[i] com os que estão à sua esquerda
        elem = seq[i]
        for j in range(i-1,-1,-1):
            if elem < seq[j]:
                # Enquanto for menor desloca os maiores uma
                   posição para a direita
                seq[j+1] = seq[j]
            else:
                # Quando não houver nenhum insere e passa ao
                   seguinte
                seq[j+1] = elem
                break
    return seq[1:]
```

Para analisar a complexidade vamos considerar o caso mais desfavorável, que ocorre quando o vector está ordenado de modo inverso. Neste caso o

ciclo interno (melhor, a comparação no interior do ciclo) é executado i vezes (de (i-1) até zero. Como o ciclo externo é executado (n-1) vezes, admitindo que o teste do **if** tem um custo c, teremos:

$$T(n) = c \times \sum_{i=2}^{n} i = c \times (\frac{n \times (n+1)}{2} - 1)$$

Daqui decorre que $T(n) = O(n^2)$.

Exercício 11.16 M

Solução

Por definição, dado uma matriz que representa um candidato a quadrado mágico, o que há a fazer é calcular o valor do número mágico e depois verificar se as linhas, colunas e diagonais somam um valor igual ao número mágico.

```
def quadrado_magico(quadrado):
  num_magico = nm(quadrado)
  # Verifica linhas
  # Verifica colunas
  # Verifica diagonais
  return resposta, num_magico
```

Uma pequena reflexão leva-nos a avançar um pouco mais. Sabemos que basta que uma das somas não seja igual ao número mágico para podermos devolver False.

```
def quadrado_magico(quadrado):
  num_magico = nm(quadrado)
  # Verifica linhas
  if not linhas(quadrado,num_magico):
    return False, num_magico
  # Verifica colunas
  elif not colunas(quadrado, num_magico):
    return False, num_magico
  # Verifica diagonais
  else:
    return diagonals(quadrate,num_magico), num_magico
```

Podemos passar agora a resolver cada um dos sub-problemas.

```
def linhas(quadrado,num_magic):
  for linha in lin(quadrado):
    if soma(linha) != num_magic:
      return False
```

```
  return True

def lin(quadrado):
  return []

def colunas(quadrado, num_magic):
  for coluna in col(quadrado):
    if soma(coluna) != num_magic:
      return False
  return True

def col(quadrado):
  return []

def diagonais(quadrado,num_magic):
  for diagonal in diag(quadrado):
    if soma(diagonal) != num_magic:
      return False
  return True

def diag(quadrado):
  return []
```

Fica por resolver como identificamos todas as linhas, colunas e diagonais. Para isso somos obrigados a clarificar a **representação** do quadrado mágico. Optamos por uma lista de listas. Identificada a representação podemos passar às funções que nos faltam.

```
def lin(quadrado):
  return quadrado

def col(quadrado):
  mat = []
  for i in range(len(quadrado)):
    linha_i = []
    for j in range(len(quadrado[0])):
      linha_i.append(quadrado[j][i])
    mat.append(linha_i)
  return mat

def diag(quadrado):
  diag_1 = []
```

```
  diag_2 = []
  for i in range(len(quadrado)):
    for j in range(len(quadrado[0])):
      if i == j:
        diag_1.append(quadrado[i][j])
      if (i+j) == (len(quadrado) - 1):
        diag_2.append(quadrado[i][j])
  return [diag_1, diag_2]


def soma(lista):
  return sum(lista)
```

O caso das linhas é trivial. No caso das colunas, feita a transposta da matriz a seguir ficamos em situação idêntica à das linhas. O caso das diagonais é mais complexo. No entanto se pensarmos que os índices dos elementos na diagonal principal são idênticos e que os índices dos elementos na outra diagonal quando somados são igual à ordem do quadrado menos um, chegamos à solução apresentada.

Falta ainda o programa que calcula o valor do número mágico. Existe uma fórmula que nos dá esse valor e daí chegamos ao programa com facilidade.

```
def nm(quadrado):
  linhas = len(quadrado)
  colunas = len(quadrado[0])
  return (linhas ** 3 + colunas) / 2
```

Exercício 11.17 M

Solução

Este problema é semelhante ao ordenamento por inserção. Apenas a estrutura de dados é mais complexa.

Exercício 11.18 M

Solução

```
def consensus(lseq):
  """
  Constrói a sequência de consenso a partir de uma lista de
     sequências de ADN
```

```
  de igual comprimento.
  """
  # 1. inicializa sequência de consenso
  cons = '' # vai ser uma string
  # 2. por cada posição da sequência
  for pos in range(len(lseq[0])): # entrada uma lista
    # 2.1 calcula qual a base mais frequente para a posição
       corrente
    # inicializa contadores das bases
    dicio_bases={'A':0,'C':0,'T':0,'G':0} # uso um dicionário
       para contar
    # para cada sequência
    for seq in lseq:
      # determina a base por cada sequência e actualiza o seu
         contador
      dicio_bases[seq[pos]]=dicio_bases[seq[pos]] + 1
    # determina a base que ocorre mais vezes
    base=max_ocorre(dicio_bases)
    # 2.2 actualiza a sequência de consenso na posição
       corrente
    cons = cons + base # a base será um caracter
  # 3. Devolve a sequência de consenso completa
  return cons


def max_ocorre(dicio):
  """A chave cujo valor associado é o maior.
  A solução depende muito do que se sabe de Python!
  Vamos ver a solução mais 'ignorante'."""
  # Vamos buscar os pares (chave, valor)
  items=list(dicio.items())
  max_val=items[0][1]
  max_ch=items[0][0]
  for par in items:
    if par[1] > max_val:
      max_val=par[1]
      max_ch=par[0]
  return max_ch
```

Os comentários no código mostram como o programa foi sendo construído e qual o racional por detrás. O programa que determina a base com maior

número de ocorrências é muito primitivo. Podia ser escrito de outros modos. eis uma alternativa possível.

```
import operator
def max_ocorre_2(dicio):
  items = list(dicio.items())
  items.sort(key=operator.itemgetter(1), reverse=True)
  return items[0][0]
```

O leitor é convidado a reflectir sobre qual das duas versões é mais interessante do ponto de vista da complexidade.

Exercício 11.19 F

Solução

Basta alterar o dicionário para poder contar com o novo símbolo.

```
dicio_bases={'A':0,'C':0,'T':0,'G':0, '-':0}
```

Podemos tornar as sequências todas com o mesmo comprimento acrescentado *gaps* à direita das mais pequenas. Com estas alterações podemos usar o programa anterior.

Exercício 11.20 MD

Solução

```
"""
Mastermind
Adivinhe o número e a posição de uma sequência de X elementos
   escolhidos
de um grupo de Y possíveis. X e Y podem variar entre 1 e 9
   sendo que Y>=X.
Comece por pedir o número de elementos diferentes existentes (
   Y) e depois o
número de elementos da sequência (X). De seguida crie uma
   sequência ordenada
de X elementos distintos, dos Y permitidos, e dê ao utilizador
    10 oportunidades
para a adivinhar. Por cada palpite do utilizador terá de lhe
   dizer quantos números
ele pôs no sítio correcto, e quantos existem na chave embora
   não nas posições
assinaladas.Guarde o histórico das jogadas e resultados para
   apresentar no ecran.
```

```
Veja o exemplo seguinte que mostra um jogo após 3 jogadas:
"""

from random import randint

def mastermind_apresenta_tabuleiro(jogadas,xx):
    """
    Apresenta o tabuleiro do jogo
    jogadas: lista com jogadas realizadas até ao momento (
       lista de listas)
    xx: quantidade de números a descobrir
    """
    print('*'*20)
    print('Tabuleiro')
    print('*'* 20)
    print('? '* xx)
    print('--'* xx)
    for jogada in jogadas:
        for nums in range(xx):
            print(str(jogada[nums]), end=' ')
        print(' pos.certa:'+str(jogada[nums+1])+' pos.errada:'
           +str(jogada[nums+2]))
    print('')


def mastermind_gera_chave(xx,yy):
    """
    Gera chave automaticamente
    xx: quantidade de números a descobrir
    yy: quantidade de valores possíveis para cada posição
    """
    chave_nova=[]
    for i in range(xx):
        a=randint(1,yy)
        while(a in chave_nova):
            a=randint(1,yy)
        chave_nova.append(a)
    return chave_nova


def mastermind_pede_jogada(tamanho_chave):
```

```
    valido=0
    while not valido:
        nova_jogada=list(input('Jogada (separe os n.os por
           vírgulas) : ').split(','))
        try:
            if len(nova_jogada)!=tamanho_chave:
                print(str(tamanho_chave)+' n.os separados por
                   vírgulas' )
            else:
                for ii in range(len(nova_jogada)):
                    nova_jogada[ii]=int(nova_jogada[ii])
                valido=1
        except:
            valido=0
            print('Erro! Deve introduzir '+str(tamanho_chave)+
               ' n.os separados por vírgulas')
    return nova_jogada


def mastermind():
    """
    Jogo do Mastermind
    """
    print('\n*** JOGO DO MASTERMIND ***\n')
    jogar = 1
    while jogar == 1:
        chave = []
        jogadas = []
        #cada jogada é guardada como uma lista de x elementos
           + elementos na posição certa + elementos existentes
            na posição errada
        x = y = 0      # x vai ter o n.o de elementos a
           descobrir, y vai ter o número de elementos
           diferentes de onde escolher
        valido = 0
        while not valido:
            try:
                x = int(input('Quantidade de números a
                   descobrir : '))
                y = int(input('Quantidade de números possíveis
                    para cada posição: '))
```

```
            if x<1 or x>9 or y<1 or y>9 or x>y:
                valido=0
                print('Erro! Deve introduzir n.o elementos
                    entre 1 e 9 e elem.possiveis>=elem.na
                   seq.')
            else:
                valido=1
        except:
            print('Erro! Deve introduzir n.o elementos
               entre 1 e 9 e elem.possiveis>=elem.na seq.'
               )
            valido=0

    #gerar chave
    chave=mastermind_gera_chave(x,y)

    #Jogo
    for i in range(10): # no máximo 10 tentativas
        #apresentar tabuleiro
        mastermind_apresenta_tabuleiro(jogadas,x)

        if i!=0 and jogadas[len(jogadas)-1][x]==x: #
           verificar se acertou e se não estamos no inicio
            break #acertou

        #pedir jogada
        nova_jogada=mastermind_pede_jogada(x)

        #avaliar jogada
        pos_certa=0  #conta números que existem na chave e
            que estão na posição certa
        pos_errada=0 #conta números que existem na chave
           mas que estão na posição errada

        for n in range(x):
            if nova_jogada[n]==chave[n]:
                pos_certa=pos_certa+1
            else:
                if nova_jogada[n] in chave:
                    pos_errada=pos_errada+1
        nova_jogada.append(pos_certa)
```

```
            nova_jogada.append(pos_errada)
            jogadas.append(nova_jogada)

        if jogadas[len(jogadas)-1][x]==x:
            print('!!! *** ACERTOU *** !!!')
        else:
            print('!!! *** EXCEDEU O N.O DE TENTATIVAS *** !!!
                ')

        # Jogar outra vez?
        resposta=input('Quer jogar outra vez (S/N)? ')
        if resposta.upper()!='S':
            jogar = 0
            print('Adeus e até à próxima!')

if __name__ == '__main__':
    mastermind()
```

Parte II

Programação Orientada aos Objectos

Capítulo 12

Tipos e Classes

Exercício 12.1 F

Solução

```
class Empty(Exception):
   """Tentativa de aceder a um contentor vazio..."""
   pass

class Stack:

   def __init__(self):
      self.stack = []

   def push(self,object):
      self.stack.insert(0,object)

   def pop(self):
      if self.is_empty():
         raise Empty('ERRO: acesso e modificação de uma pilha
            vazia!')
      return self.stack.pop(0)

   def top(self):
      if self.is_empty():
         raise Empty('ERRO: Consulta de uma pilha vazia!')
      else:
         return self.stack[0]
```

```
   def is_empty(self):
      return len(self.stack) == 0

   def len(self):
      return self.stack.__len__()

   def __str__(self):
      saida = ''
      for elem in self.stack[::-1]:
         saida = str(elem) + ',' + saida
      saida = saida[:-1]
      return '[<' + saida + ']'


if __name__ == '__main__':
   pil_1 = Stack()
   pil_1.push('A')
   print(pil_1.is_empty())
   print(pil_1)
   print(pil_1.top())
   pil_1.push('B')
   pil_1.push('C')
   print(pil_1)
   pil_1.pop()
   print(pil_1)
```

Exercício 12.2 F

Solução

```
def converte_stack(num):
   res = Stack()
   while num:
      q,r = divmod(num,2)
      num = q
      res.push(r)
   return res
```

Exercício 12.3 F

Solução

```
class Overflow(Exception):
   """Tentativa de inserir num contentor cheio..."""
   pass

class Queue_Size:

   def __init__(self,size):
      self.items = []
      self.size = size

   def insere(self, item):
      if self.items.__len__() == self.size:
         raise Overflow('ERRO: contentor cheio!')
      self.items.insert(0,item)

   def retira(self):
      if self.items == []:
         raise ValueError
      else:
         return self.items.pop()

   def consulta(self):
      if self.items == []:
         raise ValueError
      else:
         return self.items[-1]

   def len(self):
      return self.items.__len__()

   def is_empty(self):
      return self.items == []

   def __str__(self):
      saida = ''
      for elem in self.items:
         saida =  saida + ',' +  str(elem)
      saida = saida[1:]
      return '>[' + saida + ']>'
```

Exercício 12.4 M

Solução

```
class EmptyPriorityQueue(Exception):
   """Tentativa de aceder a um contentor vazio..."""
   pass

class PriorityQueue:
   """Solução básica. Os elementos (são tuplos (prioridade,
      valor) e mantidos numsa lista não ordenado."""

   def __init__(self):
      self.items = []

   def insere(self, priority,item):
      self.items.append((priority,item))

   def retira(self):
      """Element of highest priority is deleted."""
      if self.items == []:
         raise EmptyPriorityQueue('ERRO: Modificação de  Fila
            vazia.')
      else:
         return self.items.pop(self.items.index(max(self.items
            )))

   def consulta(self):
      if self.items == []:
         raise EmptyPriorityQueue('ERRO: Acesso a Fila vazia.'
            )
      else:
         return max(self.items)

   def len(self):
      return self.items.__len__()

   def is_empty(self):
      return self.items == []
```

```
   def __str__(self):
      saida = ''
      for elem in self.items:
         saida =  saida + ',' +  str(elem)
      saida = saida[1:]
      return '[' + saida + ']'

if __name__ == '__main__':
   fila_1 = PriorityQueue()
   fila_1.insere(3,'A')
   fila_1.insere(8,'B')
   fila_1.insere(5,'C')
   print(fila_1)
   print(fila_1.consulta())
   elem_1 = fila_1.retira()
   print(fila_1)
   print(fila_1.is_empty())
   print(fila_1.consulta())
   print(fila_1.len())
```

Exercício 12.5 F

Solução

```
def calcula_pos_float(exp):
   pilha_op = Stack()
   tokens = exp.split()
   for token in tokens:
      try:
         isinstance(eval(token),(int,float))
         pilha_op.push(eval(token))
      except:
         operando_2 = pilha_op.pop()
         operando_1 = pilha_op.pop()
         resultado = calcula_f(token,operando_1, operando_2)
         pilha_op.push(resultado)
   return pilha_op.pop()

def calcula_f(op,opera_1, opera_2):
   operations = {'*': lambda x, y: x * y, '+': lambda x, y: x
      + y, '-': lambda x, y: x - y, '//': lambda x, y: x // y,
```

```
      '/': lambda x, y: x / y}
   return operations[op](opera_1,opera_2)
```

Exercício 12.6 M

Solução

```
def calcula_pre_float(exp):
   pilha_op = Stack()
   tokens = exp.split()
   for token in tokens:
      try:
         if isinstance(eval(token),(int,float)) and isinstance
            (pilha_op.top(),(int,float)):
            operando_1 = pilha_op.pop()
            operando_2 = eval(token)
            op = pilha_op.pop()
            resultado = calcula_f(op,operando_1, operando_2)
            pilha_op.push(resultado)
         elif isinstance(eval(token),(int,float)):
            pilha_op.push(eval(token))
      except:
         pilha_op.push(token)
   while pilha_op.len() > 1:
      operando_2 = pilha_op.pop()
      operando_1 = pilha_op.pop()
      op = pilha_op.pop()
      resultado = calcula_f(op,operando_1, operando_2)
      pilha_op.push(resultado)
   return pilha_op.pop()

def calcula_f(op,opera_1, opera_2):
   operations = {'*': lambda x, y: x * y, '+': lambda x, y: x
      + y, '-': lambda x, y: x - y, '//': lambda x, y: x // y,
      '/': lambda x, y: x / y}
   return operations[op](opera_1,opera_2)

def main_126(exp):
   print(calcula_pre_float(exp))
```

Exercício 12.7 D

Solução

```
def infix_to_postfix(infix):
   prec = {}
   prec["*"] = 3
   prec["/"] = 3
   prec["+"] = 2
   prec["-"] = 2
   prec["("] = 1

   pilha_op = Stack()
   postfix = []

   tokens = infix.split()

   for token in tokens:
      try:
         isinstance(eval(token),(int,float))
         postfix.append(token)
      except:
         if token == '(':
            pilha_op.push(token)
         elif token == ')':
            top_token = pilha_op.pop()
            while top_token != '(':
               postfix.append(top_token)
               top_token = pilha_op.pop()
         else:
            while (not pilha_op.is_empty()) and (prec[pilha_op
               .top()] >= prec[token]):
               postfix.append(pilha_op.pop())
            pilha_op.push(token)

   while not pilha_op.is_empty():
      postfix.append(pilha_op.pop())
   return " ".join(postfix)

def main_127():
   print(infix_to_postfix("2 * 5 + 7 * 9"))
   print(infix_to_postfix("( 4.3 + 2.8 ) * 4 - ( 7.2 - 2 ) * (
       6 + 8.9 )"))
```

Exercício 12.8 F

Solução

```
class Empty(Exception):
   """Tentativa de aceder a um contentor vazio..."""
   pass

class Deque:

   def __init__(self):
      self.deque = []

   def insere_frente(self,objecto):
      self.deque.insert(0,objecto)

   def insere_tras(self,objecto):
      self.deque.append(objecto)

   def remove_frente(self):
      if self.is_empty():
         raise Empty('ERRO: acesso e modificação de uma pilha
            vazia!')
      return self.deque.pop(0)

   def remove_tras(self):
      if self.is_empty():
         raise Empty('ERRO: acesso e modificação de uma pilha
            vazia!')
      return self.deque.pop()


   def is_empty(self):
      return len(self.deque) == 0

   def len(self):
      return self.deque.__len__()

   def __str__(self):
      saida = ''
```

```
      for elem in self.deque[::-1]:
         saida = str(elem) + ',' + saida
      saida = saida[:-1]
      return '[<' + saida + '>]'


if __name__ == '__main__':
   deq_1 = Deque()
   deq_1.insere_frente('A')
   print(deq_1.is_empty())
   print(deq_1)
   deq_1.insere_frente('B')
   deq_1.insere_tras('C')
   print(deq_1)
   deq_1.remove_frente()
   print(deq_1)
```

Exercício 12.9 M

Solução

```
def capicua(cadeia):
   if cadeia.len() <= 1:
      return True
   else:
      f = cadeia.remove_frente()
      t = cadeia.remove_tras()
      return (f == t) and capicua(cadeia)
```

Exercício 12.10 F

Solução

```
import random

def create_random_vector(size, inf, sup):
   vec = Vector(size)
   for i in range(size):
      vec[i] = random.randint(inf,sup)
   return vec

def prod_escalar(v_1,v_2):
```

```
   assert len(v_1) == len(v_2), 'Devem ter o mesmo tamanho.'
   return sum([v_1[i] * v_2[i] for i in range(len(v_1))])

def main_1210(size,inf,sup):
   v_1 = create_random_vector(size,inf, sup)
   print(v_1)
   v_2 = create_random_vector(size,inf, sup)
   print(v_2)
   return prod_escalar(v_1,v_2)
```

Exercício 12.11 F

Solução

```
def translation(vector, delta):
   for i in range(len(vector)):
      vector[i] += delta
   return vector

def main_1211(size,inf,sup,delta):
   vec = create_random_vector(size,inf,sup)
   print(vec)
   return translation(vec,delta)
```

Exercício 12.12 M

Solução

```
import math

def rotate_vector_2d(vec_2d,ang):
   vec = Vector(2)
   vec[0] = vec_2d[0]* math.cos(ang) - vec_2d[1] * math.sin(
      ang)
   vec[1] = vec_2d[0]* math.sin(ang) + vec_2d[1] * math.cos(
      ang)
   return vec

def main_1212(inf,sup,ang):
   vec = create_random_vector(2,inf,sup)
   print(vec)
   return rotate_vector_2d(vec,ang)
```

Exercício 12.13 D

Solução

```
class Array2D:

   def __init__(self, numbrows, numbcols):
      self._the_rows = Vector(numbrows)

      for r in range(numbrows):
         self._the_rows[r] = Vector(numbcols)


   def numb_rows(self):
      return len(self._the_rows)

   def numb_cols(self):
      return len(self._the_rows[0])

   def clear(self, value):
      for row in self._the_rows:
         row.clear(value)

   def __getitem__(self,index):
      assert len(index) == 2, "Número de índices inválido"
      row = index[0]
      col = index[1]

      assert row >= 0 and row < self.numb_rows() and \
             col >= 0 and col < self.numb_cols(), \
             "índices fora dos limites."
      array_row = self._the_rows[row]
      return array_row[col]


   def __setitem__(self,index, value):
      assert len(index) == 2, "Número de índices inválidos"
      row = index[0]
      col = index[1]
```

```
        assert row >= 0 and row < self.numb_rows() and \
             col >= 0 and col < self.numb_cols(), \
             "índices fora dos limites"
        array_row = self._the_rows[row]
        array_row[col] = value
```

Exercício 12.14 M

Solução

```
def multiplica_matrizes(mat_1, mat_2):
   num_linhas = mat_1.numb_rows()
   num_colunas = mat_2.numb_cols()
   num_elem = mat_1.numb_cols()
   prod = Array2D(num_linhas,num_colunas)
   for i in range(num_linhas):
      for j in range(num_colunas):
         prod[i,j] = sum([mat_1[i,k] * mat_2[k,j] for k in
            range(num_elem)])
   return prod
```

Exercício 12.15 F

Solução

```
class EmptyAB(Exception):
    pass

class  ArvoreBinaria:
    """
    Uma árvore binária ou é vazia (valor = None), ou tem uma
       raiz e duas sub-árvores binárias esquerda e direita.
    A representação vai ser feita com base no conceito de nó,
       uma estrutura com três campos: um valor e dois
       ponteiros.
    """

    def __init__(self, valor=None):
        if valor:
            self.raiz = valor
            self.esquerda = ArvoreBinaria()
            self.direita = ArvoreBinaria()
```

```
        else:
            self.raiz = None

    def obtem_raiz(self):
        if self.raiz == None:
            raise EmptyAB('ERRO: Árvore Vazia!')
        return self.raiz

    def obtem_esquerda(self):
        if self.raiz == None:
            raise EmptyAB('ERRO: Árvore Vazia!')
        else:
            return self.esquerda

    def obtem_direita(self):
        if self.raiz == None:
            raise EmptyAB('ERRO: Árvore Vazia!')
        else:
            return self.direita

    def folha(self):
        if self.raiz:
            return (self.obtem_esquerda().vazia()) and (self.
               obtem_direita().vazia())
        return False

    def vazia(self):
        return self.raiz == None

    def muda_raiz(self, valor):
        if self.raiz == None:
            raise EmptyAB('ERRO: Árvore Vazia')
        self.raiz = valor

    def muda_esquerda(self,abin):
        if not isinstance(abin,ArvoreBinaria):
            raise TypeError('Não é árvore binária')
        elif self.raiz == None:
            raise EmptyAB('ERRO: Árvore Vazia')
        else:
            self.esquerda = abin
```

```
    def muda_direita(self,abin):
        if not isinstance(abin,ArvoreBinaria):
            raise TypeError('Não é árvore binária')
        elif self.raiz == None:
            raise EmptyAB('ERRO: Árvore Vazia')
        else:
            self.direita = abin

    def insere_esq(self,valor):
        if self.raiz == None:
            raise EmptyAB('ERRO: Árvore Vazia!')
        elif self.esquerda == None:
            self.esquerda = ArvoreBinaria(valor)
        else:
            temp = ArvoreBinaria(valor)
            temp.esquerda = self.esquerda
            self.esquerda = temp

    def insere_dir(self,valor):
        if self.raiz == None:
            raise EmptyAB('ERRO: Árvore Vazia!')
        elif self.direita == None:
            self.direita = ArvoreBinaria(valor)
        else:
            temp = ArvoreBinaria(valor)
            temp.direita = self.direita
            self.direita = temp

    def __str__(self, nivel=0):
        if self.raiz == None:
            return ''
        ret = ""
        # Lado direito
        if not self.direita.vazia():
            ret += self.direita.__str__(nivel + 1)
        # Raiz
        ret += "\n" + ("    "* nivel) + str(self.raiz)
        # Lado esquerdo
        if not self.esquerda.vazia():
            ret += self.esquerda.__str__(nivel + 1)
```

```
        return ret

# Travessia dem ordem
def inorder(arvore):
    if arvore.vazia():
        return []
    elif arvore.folha():
        return [arvore.obtem_raiz()]
    else:
        return inorder(arvore.obtem_esquerda()) + [arvore.
           obtem_raiz()] + inorder(arvore.obtem_direita())
```

Exercício 12.16 M

Solução

```
def elimina_no(no,ab):
    if ab.vazia():
        return ab
    elif ab.obtem_raiz() == no:
        return ArvoreBinaria()
    else:
        ab_esq = elimina_no(no, ab.obtem_esquerda())
        ab_dir = elimina_no(no,ab.obtem_direita())
        new_ab = ArvoreBinaria(ab.obtem_raiz())
        new_ab.muda_esquerda(ab_esq)
        new_ab.muda_direita(ab_dir)
        return new_ab
```

Exercício 12.17 M

Solução

```
def isomorficas(ab_1,ab_2):
   if ab_1.vazia() and ab_2.vazia():
      return True
   elif  ab_1.vazia() or ab_2.vazia():
      return False
   else:
      return isomorficas(ab_1.obtem_esquerda(),ab_2.
         obtem_esquerda()) and isomorficas(ab_1.obtem_direita
         (),ab_2.obtem_direita())
```

Exercício 12.18 D

Solução

```
class ArvoreBinariaProcura:

   def __init__(self,valor=None):
      if valor:
            self.raiz = valor
            self.esquerda = ArvoreBinariaProcura()
            self.direita = ArvoreBinariaProcura()
      else:
            self.raiz = None

   def vazia(self):
      return self.raiz == None

   def insere_esq(self,valor):
      self.insere(valor)

   def insere_dir(self,valor):
      self.insere(valor)

   def insere(self,valor):
      if self.raiz == None:
            self.raiz = ArvoreBinariaProcura(valor)
      else:
            if valor > self.raiz:
               if not self.direita.vazia():
                  self.direita.insere(valor)
               else:
                  self.direita = ArvoreBinariaProcura(valor)
            else:
               if not self.esquerda.vazia():
                  self.esquerda.insere(valor)
               else:
                  self.esquerda = ArvoreBinariaProcura(valor)


   def __str__(self, nivel=0):
```

```
        ret = ""
        # Lado direito
        if not self.direita.vazia():
              ret += self.direita.__str__(nivel + 1)
        # Raiz
        ret += "\n" + ("    "* nivel) + str(self.raiz)
        # Lado esquerdo
        if not self.esquerda.vazia():
           ret += self.esquerda.__str__(nivel + 1)
        return ret
```

Exercício 12.19 M

Solução

```
def esta_em_abp_arv(no, arv):
    if arv.vazia():
        return ArvoreBinariaProcura()
    elif arv.obtem_raiz() == no:
        return  arv
    else:
        if no < arv.obtem_raiz():
            return esta_em_abp_arv(no,arv.obtem_esquerda())
        else:
            return esta_em_abp_arv(no, arv.obtem_direita())
```

Exercício 12.20 M

Solução

```
def esta_em_abp(no, arv):
   if arv.vazia():
      return False
   elif arv.obtem_raiz() == no:
      return True
   else:
      if no < arv.obtem_raiz():
            return esta_em_abp(no,arv.obtem_esquerda())
      else:
            return esta_em_abp(no, arv.obtem_direita())
```

Capítulo 13

Programação Orientada aos Objectos

Exercício 13.1 MF

Solução

Exercício 13.2 F

Solução

```
class Ponto:

    def __init__(self):
        self._x = 0
        self._y = 0

    def def_coordenadas(self, x, y):
        self._x = x
        self._y = y
```

Indicamos apenas as mudanças necessárias à solução inicial.

Exercício 13.3 F

Solução

```
class Ponto:

    def __init__(self, x=0, y=0):
        self._x = x
        self._y = y
```

Apresentamos apenas a mudança necessária no construtor. O resto é igual ao primeiro exemplo.

Exercício 13.4 F

Solução

Exercício 13.5 M

Solução

Exercício 13.6 M

Solução

Exercício 13.7 M

Solução

```
class Contador:
    contador = 0
    def __init__(self,valor=0):
        Contador.contador = valor

    def conta_mais(self):
        Contador.contador += 1

    def conta_menos(self):
        if Contador.contador:
            Contador.contador -= 1
        else:
            raise ValueError('ERRO: impossível decrementar o
               contador!')

    def reset(self, valor=0):
        Contador.contador = valor

    def valor_contador(self):
        return Contador.contador
```

Exercício 13.8 M

Solução

Exercício 13.9 M

Solução

Exercício 13.10 F

Solução

```
import math

class Circulo:

    def __init__(self,raio):
        self.raio = raio

    def area(self):
        return math.pi * self.raio**2

    def __str__(self):
        return "%s" % self.area()

class Cilindro():

    def __init__(self,raio,altura):
        self.base = Circulo(raio)
        self.altura = altura

    def volume(self):
        return self.base.area() * self.altura

    def __str__(self):
        return "%s" % self.volume()

class Cone(Cilindro):

    def __init__(self,raio,altura):
        super().__init__(raio,altura)

    def volume(self):
```

```
        return self.base.area() * self.altura / 3

def main_1310():
    circo = Circulo(10)
    print(circo)

    cilindro = Cilindro(10, 2)
    print(cilindro)

    cone = Cone(10,2)
    print(cone)
```

Exercício 13.11 F

Solução

```
class ContaBancaria:

    numero_conta = 0

    def __init__(self,nome, entrada=0):
        self.nome = nome
        self.saldo = entrada
        ContaBancaria.numero_conta += 1
        self.numero = ContaBancaria.numero_conta

    def manutencao(self):
        raise NotImplementedError('ERRO: método não
           implementado!')

    def get_saldo(self):
        return self.saldo

class ContaOrdem(ContaBancaria):

    def __init__(self,nome,entrada=0):
        super().__init__(nome,entrada)

    def manutencao(self):
        if self.saldo < 10**3:
            self.saldo -= 0.02
```

```
        else:
            self.saldo -= 0.01




class ContaPrazo(ContaBancaria):

    def __init__(self,nome,entrada=0):
        super().__init__(nome,entrada)

    def manutencao(self):
        self.saldo -= 0.005



class ContaPoupanca(ContaBancaria):

    def __init__(self,nome,entrada=0):
        super().__init__(nome,entrada)

    def manutencao(self):
        if self.saldo < 10**4:
            self.saldo -= 0.01

def main_1311():
    cordem = ContaOrdem('EC',100)
    cprazo = ContaPrazo('AB',1000)
    cpoupa = ContaPoupanca('DC',5000)

    print(cordem.get_saldo())
    print(cprazo.get_saldo())
    print(cpoupa.get_saldo())

    cordem.manutencao()
    cprazo.manutencao()
    cpoupa.manutencao()

    print(cordem.get_saldo())
    print(cprazo.get_saldo())
```

```
    print(cpoupa.get_saldo())
```

Exercício 13.12 F

Solução

```
class Mundo:

    def __init__(self, tamanho):
        self._tamanho = tamanho
        self._grelha = [['*'] * tamanho for i in range(tamanho
           )]

    def mostra_mundo(self):
        print('-' * self._tamanho * 7)
        print()
        for i in range(self._tamanho):
            linha = self._grelha[i]
            for j in range(self._tamanho):
                print('%-6s'% linha[j] + '  ', end='')
            print()
            print()
        print('-' * self._tamanho * 7)
        print()

    def regista(self, robot):
        x,y = robot.obtem_posicao()
        self._grelha[y][x] = robot

    def limpa_registo(self, x,y):
        self._grelha[y][x] = '*'


    def obtem_tamanho(self):
        return self._tamanho

    def celula_vazia(self, x, y):
        return self._grelha[y][x] == '*'

    def obtem_conteudo(self, pos_x, pos_y):
        return self._grelha[pos_y][pos_x]
```

```
class Robot:

    def __init__(self, nome, mundo, pos_x=0, pos_y=0):
        self._nome = nome
        self._mundo = mundo
        self._pos_x = pos_x
        self._pos_y = pos_y

    def obtem_nome(self):
        return self._nome

    def obtem_mundo(self):
        return self._mundo

    def obtem_posicao(self):
        return self._pos_x, self._pos_y

    def define_posicao(self, x,y):
        self._pos_x = x
        self._pos_y = y

    def move(self):
        from random import choice
        delta = [(-1,0),(1,0),(0,-1),(0,1),(-1,1),(1,1)
           ,(-1,-1),(1,-1)] # Modificação
        x,y = self.obtem_posicao()
        d_x, d_y = choice(delta)
        n_x =  x + d_x
        n_y = y + d_y
        tamanho = self._mundo.obtem_tamanho()
        if (0 <= n_x < tamanho) and (0 <= n_y < tamanho):
            if self._mundo.celula_vazia(n_x,n_y):
                self._mundo.limpa_registo(x,y)
                self.define_posicao(n_x,n_y)
                self._mundo.regista(self)
                self._energia -= 1
            else:
                print('Célula Ocupada')
        else:
```

```
            print('Movimento Impossível.')



    def __str__(self):
        return  self._nome

    def obtem_energia(self):
        return self._energia


class Predador(Robot):

    def __init__(self, nome, mundo, pos_x=0, pos_y=0, energia
       =100):
        super().__init__(nome, mundo,pos_x, pos_y)
        self._energia = energia

    def come(self):
        vizinhos = [(-1,0),(1,0),(0,-1),(0,1),(-1,1),(1,1)
           ,(-1,-1),(1,-1)] # Modificação
        x,y = self.obtem_posicao()
        for d_x, d_y in vizinhos:
            n_x =  x + d_x
            n_y = y + d_y
            tamanho = self._mundo.obtem_tamanho()
            if (0 <= n_x < tamanho) and (0 <= n_y < tamanho):
                conteudo = self._mundo.obtem_conteudo(n_x,n_y)
                if isinstance(conteudo,Presa):
                    self._mundo.limpa_registo(x,y)
                    self.define_posicao(n_x,n_y)
                    self._mundo.regista(self)
                    self._energia += conteudo.obtem_energia()
                    break
                else:
                    continue
        else:
            print('Impossível comer.')
```

```
class Presa(Robot):
    def __init__(self, nome, mundo, pos_x=0, pos_y=0, energia
       =100):
        super().__init__(nome, mundo,pos_x, pos_y)
        self._energia = energia



def simula(n,tamanho):
    from random import randint
    m = Mundo(tamanho)
    r1 = Predador('PD1', m, tamanho//2,tamanho//2)
    m.regista(r1)
    r2 = Presa('PS1',m, tamanho//2 + 1, tamanho//2)
    m.regista(r2)
    r3 = Presa('PS2', m, randint(0,tamanho-1), randint(0,
       tamanho-1))
    m.regista(r3)
    m.mostra_mundo()

    for i in range(n):
        r1.come()
        r1.move()
        m.mostra_mundo()
    print(r1.obtem_energia())
```

Exercício 13.13 M

Solução

Exercício 13.14 M

Solução

```
def move(self):
    from random import choice
    delta = [(-1,0),(1,0),(0,-1),(0,1)]
    x,y = self.obtem_posicao()
    d_x, d_y = choice(delta)
    n_x =  x + d_x
    n_y = y + d_y
```

```
    tamanho = self._mundo.obtem_tamanho()
    while not ((0 <= n_x < tamanho) and (0 <= n_y < tamanho)
       and self._mundo.celula_vazia(n_x,n_y)):
        d_x, d_y = choice(delta)
        n_x =  x + d_x
        n_y = y + d_y
    self._mundo.limpa_registo(x,y)
    self.define_posicao(n_x,n_y)
    self._mundo.regista(self)
```

Nesta solução há o perigo de o programa entrar em ciclo caso não exista nenhuma célula vazia na vizinhança do robô.

Exercício 13.15 M

Solução

```
def move(self):
    from random import choice
    delta = [(-1,0),(1,0),(0,-1),(0,1)]
    x,y = self.obtem_posicao()
    tamanho = self._mundo.obtem_tamanho()
    d_x, d_y = choice(delta)
    n_x =  (x + d_x) % tamanho
    n_y = (y + d_y) % tamanho

    if self._mundo.celula_vazia(n_x,n_y):
        self._mundo.limpa_registo(x,y)
        self.define_posicao(n_x,n_y)
        self._mundo.regista(self)
    else:
        print('Célula Ocupada')
```

Como se pode ver as alterações são mínimas. Basta calcular a nova posição módulo o tamanho do mundo.

Exercício 13.16 M

Solução

```
class Predador(Robot):

    def __init__(self, nome, mundo, pos_x=0, pos_y=0, energia
       =100):
```

```
        super().__init__(nome, mundo,pos_x, pos_y)
        self._energia = energia

    def move(self):
        from random import choice
        delta = [(-1,0),(1,0),(0,-1),(0,1)]
        x,y = self.obtem_posicao()
        tamanho = self.obtem_mundo().obtem_tamanho()
        if self._energia <= 0: # morre??
            self._mundo.limpa_registo(x,y)
            return
        # Procura uma presa
        for d_x, d_y in delta:
            n_x =  (x + d_x) % tamanho
            n_y =  (y + d_y) % tamanho
            conteudo = self._mundo.obtem_conteudo(n_x,n_y)
            if isinstance(conteudo,Presa):
                energia_mais = conteudo.obtem_energia()
                self._mundo.limpa_registo(x,y)
                self.define_posicao(n_x,n_y)
                self._mundo.regista(self)
                self._energia += energia_mais - 10
                break
            else:
                continue
        else: # não encontrou presa
            # procura célula vazia
            for d_x,d_y in delta:
                n_x =  (x + d_x) % tamanho
                n_y =  (y + d_y) % tamanho
                if self._mundo.celula_vazia(n_x,n_y):
                    self._mundo.limpa_registo(x,y)
                    self.define_posicao(n_x,n_y)
                    self._mundo.regista(self)
                    self._energia -= 10
                    break
                else:
                    continue
            # nem presa nem célula vazia
            else:
                print('movimento impossível')
```

```
class Presa(Robot):
    def __init__(self, nome, mundo, pos_x=0, pos_y=0, energia
       =100):
        super().__init__(nome, mundo,pos_x, pos_y)
        self._energia = energia


    def move(self):
        vizinhos = [(0,-1),(1,0),(0,1),(-1,0)]
        tamanho = self.obtem_mundo().obtem_tamanho()
        x,y = self.obtem_posicao()
        if self._energia <= 0: # morre??
            self._mundo.limpa_registo(x,y)
            return
        for d_x, d_y in vizinhos:
            n_x =  (x + d_x) % tamanho
            n_y = (y + d_y) % tamanho
            tamanho = self._mundo.obtem_tamanho()
            conteudo = self._mundo.obtem_conteudo(n_x,n_y)
            if isinstance(conteudo,Predador):
                continue
            elif isinstance(conteudo,Presa):
                continue
            else:
                self._mundo.limpa_registo(x,y)
                self.define_posicao(n_x,n_y)
                self._mundo.regista(self)
                self._energia -= 10
                break
        else:
            print('Movimento Impossível.')
```

Como se pode ver alterámos as definições de movimento para cada uma das duas classes de robots. O Predador procura primeiro uma presa. A Presa foge para uma célula vizinha não ocupada. Com a definição de vizinhança

que estamos a usar é garantido que nesta nova posição não será vista.

Exercício 13.17 D

Solução

Exercício 13.18 MD

Solução

Exercício 13.19 M

Solução

```
class ContaBancaria:

    numero_conta = 0

    def __init__(self,nome, entrada=0):
        self.nome = nome
        self.saldo = entrada
        ContaBancaria.numero_conta += 1
        self.numero = ContaBancaria.numero_conta

    def manutencao(self):
        raise NotImplementedError('ERRO: método não
           implementado!')

    def get_saldo(self):
        return self.saldo

    def deposito(self, montante):
        self.saldo += montante

class ContaOrdem(ContaBancaria):

    def __init__(self,nome,entrada=0):
        super().__init__(nome,entrada)

    def manutencao(self):
        if self.saldo < 10**3:
```

```
            self.saldo -= 0.02
        else:
            self.saldo -= 0.01

    def levantamento(self,montante):
        if self.saldo - montante < 0:
            print('Operação impossível: Saldo insuficiente.')
        else:
            self.saldo -= montante

class ContaPrazo(ContaBancaria):

    def __init__(self,nome,entrada=0):
        super().__init__(nome,entrada)

    def manutencao(self):
        self.saldo -= 0.005

    def levantamento(self, montante):
        if self.saldo - montante < 10**3:
                    print('Operação impossível: Saldo
                       insuficiente.')
        else:
            self.saldo -= 1.001 * montante

class ContaPoupanca(ContaBancaria):

    def __init__(self,nome,entrada=0):
        super().__init__(nome,entrada)

    def manutencao(self):
        if self.saldo < 10**4:
            self.saldo -= 0.01

    def levantamento(self, montante):
        print('Operação não autorizada.')

def main_1319():
    cordem = ContaOrdem('EC',100)
    cprazo = ContaPrazo('AB',10000)
    cpoupa = ContaPoupanca('DC',5000)
```

```
    print(cordem.get_saldo())
    print(cprazo.get_saldo())
    print(cpoupa.get_saldo())

    cordem.deposito(200)
    cprazo.levantamento(500)
    cpoupa.levantamento(1000)

    print(cordem.get_saldo())
    print(cprazo.get_saldo())
    print(cpoupa.get_saldo())
```

Exercício 13.20 M

Solução

Exercício 13.21 MD

Solução

```
"""
Simulador de Circuitos lógicos.
PB 13.21
Ernesto Costa
"""

# -----------  Classe Conector  ----------------
class Conector:
    """
    Os conectores podem ser de entrada ou de saída.
    Caso seja mudada a saída deve ser propagado para as
       entradas a que
    está ligado.
    """

    def __init__(self,nome, proprietario, activa=False):
        """
        @activa: está ligado ou "no ar"
        """
        self._nome = nome
```

```
        self._proprietario = proprietario
        self._activa = activa
        self._valor = None
        self._ligados = []


    def obtem_nome(self):
        return self._nome

    def obtem_propietario(self):
        return self._proprietario

    def obtem_activa(self):
        return self._activa

    def obtem_valor(self):
        return self._valor

    def liga(self,entradas):
        if not isinstance(entradas,list):
            entradas = [entradas]
        for entrada in entradas:
            self._ligados.append(entrada)

    def define_valor(self, valor):
        if self._valor != valor:
            self._valor = valor
            if self._activa:
                self._proprietario.avalia()
            for ligado in self._ligados:
                ligado.define_valor(valor)

# -------- Classe Circuito Lógico ---------------
class CircuitoLogico:
    """ Apenas definem nome e abstraiem a função de avaliação.
       """

    def __init__(self,nome):
        self._nome = nome

    def obtem_nome(self):
```

```
        return self._nome

    def avalia(self):
        return


# --------  Portas Lógicas ------------
# --- Binárias
class PortaBinaria(CircuitoLogico):

    def __init__(self, nome):
        super().__init__(nome)
        self._ent_A = Conector('A',self,1)
        self._ent_B = Conector('B',self,1)
        self._saida_C = Conector('A',self)

    def __str__(self):
        return '[' + str(self._ent_A) + ' : ' + str(self.
           _ent_B) + ' = ' + str(self._saida_C) + ']'

# --- Unária: apenas o NOT pelo que não se justifica abstrair
   para portas unárias...
class NOT(CircuitoLogico):

    def __init__(self, nome):
        super().__init__(nome)
        self._ent_A = Conector('A',self,1)
        self._saida_B = Conector('B',self)

    def avalia(self):
        self._saida_B.define_valor(not self._ent_A.obtem_valor
           ())

    def __str__(self):
        return '[' + str(self._ent_A) +  ' = ' + str(self.
           _saida_B) + ']'

class AND(PortaBinaria):

    def __init__(self,nome):
        super().__init__(nome)
```

```
    def avalia(self):
        self._saida_C.define_valor(self._ent_A.obtem_valor()
           and self._ent_B.obtem_valor())


class OR(PortaBinaria):

    def __init__(self,nome):
        super().__init__(nome)

    def avalia(self):
        self._saida_C.define_valor(self._ent_A.obtem_valor()
           or self._ent_B.obtem_valor())


# ---- Aux ------
def bit_to_true_value(cad_bin,pos):
    """ Devolve o valor de verdade do bit da cvadeia binária
       na posição pos.
    >>> bit_to_true_value('10101',3)
    False
    """
    return cad_bin[pos] == '1'

def true_value_to_bit(true_value):
    if true_value:
        return '1'
    else:
        return '0'


# ------ Circuitos ----

class Circuito1(CircuitoLogico):
    """ Circuito da figura 13.16."""
    def __init__(self,nome):
        super().__init__(nome)
        self.A = Conector('A',self,1)
        self.B = Conector('B',self,1)
        self.C = Conector('C',self,1)
```

```
        self.D = Conector('D',self,1)
        self.S = Conector('S',self)

        self.A1 = AND('A1')
        self.A2 = AND('A2')
        self.O1 = OR('O1')
        self.N1 = NOT('N1')

        self.A.liga(self.A1._ent_A)
        self.B.liga(self.A1._ent_B)
        self.C.liga(self.A2._ent_A)
        self.D.liga(self.A2._ent_B)

        self.A1._saida_C.liga(self.O1._ent_A)
        self.A2._saida_C.liga(self.O1._ent_B)

        self.O1._saida_C.liga(self.N1._ent_A)

        self.N1._saida_B.liga(self.S)

    def define_entrada(self,entrada):
        assert len(entrada) == 4, 'ERRO: a entrada tem que ter
            4 bits...'
        A = bit_to_true_value(entrada,0)
        B = bit_to_true_value(entrada,1)
        C = bit_to_true_value(entrada,2)
        D = bit_to_true_value(entrada,3)

        self.A.define_valor(A)
        self.B.define_valor(B)
        self.C.define_valor(C)
        self.D.define_valor(D)

    def obtem_saida(self):
        return self.S.obtem_valor()


def main_circo_1(entrada):
    assert len(entrada) == 4, 'ERRO: a entrada tem que ter 4
       bits...'
```

```
    circo = Circuito1('meu_circo')
    circo.define_entrada(entrada)
    return circo.obtem_saida()



if __name__ == '__main__':
    # testar para todas as entradas possíveis
    for i in range(16):
        entrada = '{0:04b}'.format(i)
        #print(entrada)
        saida = true_value_to_bit(main_circo_1(entrada))
        print('ENTRADA: %s\nSAÍDA: %s' % (entrada,saida))
        print()
```

Capítulo 14

Interfaces Gráficas com o Utilizador

Exercício 14.1 MF

Solução

```
# ------------------ Variante com grid
raiz = Tk()
raiz.title('Exemplo Básico')



quadro = Frame(raiz)
quadro.grid()

etiqueta1 = Label(quadro,text='Primeiro')
etiqueta1.grid(row=0,column=0)

nome_entrada_1=StringVar()
nome_entrada_1.set('')

nome_p = Entry(quadro,textvariable=nome_entrada_1)
nome_p.grid(row=0,column=1)

nome_entrada_2=StringVar()
nome_entrada_2.set('')

nome_s = Entry(quadro,textvariable=nome_entrada_2)
nome_s.grid(row=1,column=0)

etiqueta2 = Label(quadro,text='Segundo')
```

```
etiqueta2.grid(row=1,column=1)

raiz.mainloop()

# ------------------ Variante com pack
raiz= Tk()
raiz.title('Contador')

quadro1 = Frame(raiz)
quadro1.pack()

etiqueta1 = Label(quadro1, text='Primeiro')
etiqueta1.pack(side=LEFT)
entrada1 = Entry(quadro1)
entrada1.pack(side=LEFT)

quadro2 = Frame(raiz)
quadro2.pack()

entrada2 = Entry(quadro2)
entrada2.pack(side=LEFT)
etiqueta2 = Label(quadro2,text='Segundo')
etiqueta2.pack(side=LEFT)


raiz.mainloop()
```

Exercício 14.2 MF

Solução

```
raiz = Tk()
raiz.title('Botões')
raiz.geometry('200x50')

quadro = Frame(raiz)
quadro.pack()

botao1 = Button(quadro,text='OK',width=10)
botao1.pack(side=LEFT)
```

```
botao2 = Button(quadro,text='KO',width=10)
botao2.pack()
raiz.mainloop()
```

Exercício 14.3 F

Solução

```
from tkinter import *

class Calcula_velocidade:

    def __init__(self,janela):

        self.entrada1=Entry(janela,width=25, bg='yellow')
        self.entrada1.pack(fill=BOTH)

        self.entrada2=Entry(janela,width=25, bg='yellow')
        self.entrada2.bind("<Return>",self.__calcula)
        self.entrada2.pack(fill=BOTH)

        self.formula=Label(janela)
        self.formula.pack()

        self.butao=Button(janela,text='Limpa', command=self.
            __limpa)
        self.butao.pack()


    def __calcula(self,evento):
        # Tem protecção de erros.
        while True:
            valor1 = self.entrada1.get()
            valor2 = self.entrada2.get()
            try:
                res = float(valor1)/float(valor2)
            except NameError:
                self.formula.configure(text="*** ERRO ***:
                    Entre de novo a formula sff. ")
                self.__limpa()
            else:
```

```
                break
        self.formula.configure(text="Resultado = " + str(res))

    def __limpa(self):
        self.entrada1.delete(0,END)
        self.entrada2.delete(0,END)
        self.formula.configure(text="")



if __name__ == '__main__':
    janela=Tk()
    janela.title("Calcula Velocidade")
    calc=Calcula_velocidade(janela)
    mainloop()
```

Exercício 14.4 F

Solução

```
raiz = Tk()
raiz.title('Botões')
raiz.geometry('200x50')

quadro = Frame(raiz)
quadro.pack()

etiqueta = Label(quadro)
etiqueta.pack()

def _ok_press():
    etiqueta['fg'] = 'green'
    etiqueta['text'] = 'Botão OK pressionado'

def _ko_press():
    etiqueta['fg'] = 'red'
    etiqueta['text'] = 'Botão KO pressionado'

botao1 = Button(quadro,text='OK',width=10, command=_ok_press)
botao1.pack(side=LEFT)
```

```
botao2 = Button(quadro,text='KO',width=10,command=_ko_press)
botao2.pack()


raiz.mainloop()
```

Exercício 14.5 F

Solução

```
import sys

from tkinter import *

class Quadro:

  def __init__(self):
    self.raiz=Frame(width=100,height=100,bg='light gray')
    self.raiz.pack(expand=YES,fill=BOTH)

    self.can=Canvas(self.raiz,bg='green')
    self.can.pack(expand=YES,fill=BOTH)

    self.entrada=Entry(self.raiz,fg='white',bg='blue')
    self.entrada.pack(expand=YES,fill=BOTH)

    self.etiqueta=Label(self.raiz,text='Valor',bg='yellow', fg
       ='magenta')
    self.etiqueta.pack(expand=YES,fill=BOTH)

    self.lista=Listbox(self.raiz,bg='light blue')
    self.lista.pack()

    # Problemas com a cor dos botoes...
    self.botao=Button(self.raiz,text='Ola')
    self.botao.pack(expand=YES,fill=BOTH)
    self.botao.configure(bg='red')

    self.botaoradial=Radiobutton(self.raiz,text='Escolha')
    self.botaoradial.pack(expand=YES,fill=BOTH)
```

```
    self.botaocheck=Checkbutton(self.raiz,text='On')
    self.botaocheck.pack(expand=YES,fill=BOTH)

    self.botao=Button(self.raiz,text='Sair', command=self.raiz
       .quit)
    self.botao.pack(expand=YES,fill=BOTH)

    def quit(self):
      sys.exit()



    self.raiz.mainloop()


if __name__ == '__main__':
  Quadro()
```

Exercício 14.6 F

Solução

```
raiz = Tk()
raiz.title('Vencimento')


quadro1 = Frame(raiz)
quadro1.grid()

horas = Label(quadro1,text='Horas')
horas.grid(row=0,column=0)

horas_ent = Entry(quadro1)
horas_ent.grid(row=0,column=1)

sal_h = Label(quadro1,text='Sal. Hora')
sal_h.grid(row=1,column=0)

sal_h_ent = Entry(quadro1)
sal_h_ent.grid(row=1,column=1)
```

```
quadro2 = Frame(raiz)
quadro2.grid()

bruto = Label(quadro2,text='V. Bruto')
bruto.grid(row=0,column=0)

bruto_var = StringVar()
bruto_var.set('0')

bruto_ent = Entry(quadro2, textvariable=bruto_var)
bruto_ent.grid(row=0,column=1)

descontos_var = StringVar()
descontos_var.set('0')

descontos = Label(quadro2,text='Descontos')
descontos.grid(row=1,column=0)

desc_ent = Entry(quadro2, textvariable=descontos_var)
desc_ent.grid(row=1,column=1)

liquido = Label(quadro2,text='V. Líquido')
liquido.grid(row=2,column=0)

liquido_var = StringVar()
liquido_var.set('0')

liquido_ent = Entry(quadro2, textvariable=liquido_var)
liquido_ent.grid(row=2,column=1)


def _calcula():
    hr = horas_ent.get()
    vh = sal_h_ent.get()
    bruto_var.set(str(float(hr)*float(vh)))
    descontos_var.set(str(float(bruto_var.get()) * 0.2))
    liquido_var.set(str(float(bruto_var.get()) - float(
       descontos_var.get())))
```

```
botao1 = Button(quadro2,text='Calcula',command=_calcula)
botao1.grid(row=3,column=0)

def _limpa():
    horas_ent.delete(0,END)
    sal_h_ent.delete(0,END)

    bruto_var.set('0')
    descontos_var.set('0')
    liquido_var.set('0')


botao2 = Button(quadro2,text='Limpa',command=_limpa)
botao2.grid(row=3,column=1)



mainloop()
```

Exercício 14.7 M

Solução

```
from tkinter import *

def grelha(n,m):
    for y in range(n):
        for x in range(m):
            botao = Button(quadro,text="(%d,%d)" % (x,y))
            botao.grid(row=y,column=x)

raiz = Tk()
raiz.title('Usar Grid')

quadro = Frame(raiz)
quadro.grid()

grelha(4,7)

raiz.mainloop()
```

Exercício 14.8 M

Solução

```
janela=Tk()
janela.title('Ficha Pessoal')
janela['bg']='light yellow'

# Texto
texto1=Label(janela,text='Nome',bg='light yellow')
texto2=Label(janela,text='Morada',bg='light yellow')
texto3=Label(janela,text='Cargo',bg='light yellow')

# Entry
entrada1=Entry(janela)
entrada2=Entry(janela)
entrada3=Entry(janela)

# Imagem
can=Canvas(janela,width=150,height=180, bg='white')
foto=PhotoImage(file='espadas.gif')
item=can.create_image(73,97,image=foto)

# Posiciona os widgets
texto1.grid(row=0, sticky=W)
texto2.grid(row=1,sticky=W)
texto3.grid(row=2,sticky=W)

entrada1.grid(row=0,column=1)
entrada2.grid(row=1,column=1)
entrada3.grid(row=2,column=1)

can.grid(row=0,column=2,rowspan=3,padx=10,pady=10)

janela.mainloop()
```

Exercício 14.9 M

Solução

```
from random import randint
```

```
def quadrado(x,y,lado,cor='black'):
    """Cria um quadrado."""
    can.create_rectangle(x,y,x+lado,y+lado,fill=cor)



janela = Tk()
janela.title('Tabuleiro')
can = Canvas(janela,width=160,height=160)
can.pack(side=TOP,padx=5,pady=5)

def tabuleiro():
    can.delete(ALL)
    for lin in range(8):
        # desenha linha
        for col in range(8):
            if (lin + col)% 2 == 0:
                quadrado(20*lin,20*col,20,'gray')
            else:
                quadrado(20*lin,20*col,20,'black')

botao_1 = Button(janela,text='Tabuleiro',command=tabuleiro)
botao_1.pack(fill=BOTH)


janela.mainloop()
```

Exercício 14.10 M

Solução

```
from tkinter import *
from random import randint

def quadrado(x,y,lado,cor='black'):
  """Cria um quadrado."""
  can.create_rectangle(x,y,x+lado,y+lado,fill=cor)


def tabuleiro():
```

```
  """Desenha o tabuleiro. Quadrado a quadrado"""
  can.delete(ALL)
  for lin in range(8):
    # desenha linha
    for col in range(8):
      if (lin + col)% 2 == 0:
        quadrado(20*lin,20*col,20,'gray')
      else:
        quadrado(20*lin,20*col,20,'black')

def piao():
  """ Desenha um pião num quadrado de um tabuleiro."""
  x = randint(0,7)
  y = randint(0,7)
  can.create_oval(x*20,y*20,(x+1)*20,(y+1)*20,fill='white')



if __name__ == '__main__':
  janela = Tk()
  janela.title('Tabuleiro com Piões')
  can = Canvas(janela,width=160,height=160)
  can.pack(side=TOP,padx=5,pady=5)

  butao_1 = Button(janela,text='Tabuleiro',command=tabuleiro)
  butao_1.pack(side=LEFT,padx=3,pady=3)

  butao_2 = Button(janela,text='Pião',command=piao)
  butao_2.pack(side=RIGHT,padx=3,pady=3)

  janela.mainloop()
```

Exercício 14.11 M

Solução

```
from tkinter import *

raiz= Tk()
raiz.title('Cores')
```

```
quadro1 = Frame(raiz)
quadro1.pack()

etiqueta1 = Label(quadro1, text='Cor')
etiqueta1.pack(side=LEFT)

cor = StringVar()
cor.set('gray')

entrada = Entry(quadro1, textvariable=cor)
entrada.pack(side=LEFT)

def muda_cor(evento):
  tela.create_rectangle(0,0,200,200,fill=cor.get())

entrada.bind('<Return>', muda_cor)
quadro2 = Frame(raiz)
quadro2.pack()

tela = Canvas(quadro2, height=200,width=200, bg=cor.get())
tela.pack(side=LEFT)


raiz.mainloop()
```

Exercício 14.12 M

Solução

```
raiz = Tk()
raiz.title('Texto que se move...')

tela = Canvas(raiz,width=300,height=50)
tela.pack(fill=BOTH)

sg = IntVar()
sg.set(1)

def stop_and_go():
    global sg
    if sg.get() == 1:
```

```
        sg.set(0)
    else:
        sg.set(1)

stop_go = Button(raiz,text='STOP/GO',command=stop_and_go)
stop_go.pack()

largura_tela = 300
delta_x = 3
x = 0
tela.create_text(x,25, text='Ernesto Costa', tags='text')

while True:
    if sg.get() == 1:
        tela.move('text',delta_x, 0)
        tela.after(100)
        tela.update()
        if x < largura_tela:
            x += delta_x
        else:
            x = 0
            tela.delete('text')
            tela.create_text(x,25, text='Ernesto Costa', tags=
               'text')
    else:
        raiz.wait_variable(sg)

raiz.mainloop()
```

Exercício 14.13 M

Solução

```
raiz = Tk()
raiz.title('Conversor Temperaturas')
raiz.grid()

quadro = Frame(raiz)
quadro.grid()

etiqueta = Label(quadro,text='Entre temperatura')
```

```
etiqueta.grid(row=0, columnspan=2)

temp = StringVar()
temp.set('')

entrada = Entry(quadro,textvariable=temp)
entrada.grid(row=1, columnspan=2)

def f_to_c():
    f = eval(temp.get())
    c = round((f-32)*5/9,2)
    temp.set(str(c))

botao1 = Button(quadro,text='F -> C', command=f_to_c)
botao1.grid(row=2,column=0)


def c_to_f():
    c = eval(temp.get())
    f = round((9/5 * c ) +32,2)
    temp.set(str(f))

botao2 = Button(quadro,text='C -> F', command=c_to_f)
botao2.grid(row=2,column=1)

raiz.mainloop()
```

Exercício 14.14 M

Solução

Exercício 14.15 F

Solução

Exercício 14.16 M

Solução

Exercício 14.17 M

Solução

Exercício 14.18 M

Solução

Exercício 14.19 D

Solução

```
from tkinter import *

class Calculadora(Frame):


  def __init__(self):
    Frame.__init__(self)
    self.pack(expand=YES,fill=BOTH)
    self.master.title('Calculadora Simples')
    self.master.iconname('calc')

    display=StringVar()
    Entry(self.master,textvariable=display).pack(side=LEFT,
       expand=YES,fill=BOTH)

    for key in ('123','456','789','-0.'):
      keyF=self.frame(self,TOP)
      for char in key:
        self.button(keyF,LEFT,char,
        lambda w=display, s='%s'% char: w.set(w.get() + s))

    opsF=self.frame(self,TOP)
    for char in '+-*/=':
      if char =='=':
        btn=self.button(opsF,LEFT,char)
        btn.bind('<ButtonRelease-1>',
        lambda e, s=self,w=display: s.calc(w))
```

```
      else:
        btn=self.button(opsF,LEFT,char,
        lambda w=display, c=char: w.set(w.get()+c))
    clearF=self.frame(self,BOTTOM)
    self.button(clearF,LEFT,'Limpa',lambda w=display: w.set('')
       ))

  def calc(self,display):
    try:
      display.set(eval(display.get()))
    except ValueError:
      display.set('ERRO')

  @staticmethod
  def frame(root,side):
    w=Frame(root)
    w.pack(side=side,expand=YES,fill=BOTH)
    return w

  @staticmethod
  def button(root,side,text,command=None):
    w=Button(root,text=text,command=command)
    w.pack(side=side,expand=YES,fill=BOTH)
    return w



if __name__ == '__main__':
    Calculadora()
    mainloop()
```

Exercício 14.20 D

Solução

Exercício 14.21 MD

Solução

A solução apresentada foi criada por Paulo Marques, actualmente CTO da empresa Feedzai.

```
from tkinter import Canvas, Frame
import time
import sys

#-----------

def hanoi(n, a, b, tmp, move=(lambda a, b: sys.stdout.write('
   Moving 1 disk from %s to %s\n' % (a, b)))):
    """Hanoi Tower algorithm. Moves <n> disks from <a> to <b>
       using <tmp> as auxiliary.
       (It doesn't really matter what <a>, <b> and <tmp> are.
          It can be numbers, strings, etc.)
       <move> represents a callback funcion. Everytime a move
          is made this function is invoked as
       move(from, to). <from> and <to> correspond to either <a
          >, <b> or <tmp>. If no <move> is
       provided, hanoi() prints out to the console the list of
           moves."""

    if n==0:
        return

    hanoi(n-1, a, tmp, b, move)
    move(a, b)
    hanoi(n-1, tmp, b, a, move)

#---------

class Hanoi(Frame):
    """Class that implements the programs GUI"""

    delay  = 0.8        # Delay between moves
    width  = 500       # Window width
    height = 200        # Window height
    sep    = 30         # Major separation among items
    dx     = 5          # Small separation among items

    #-----------
    def __init__(self, n_disks):
        """Class constructor. Receives number of disks as
           parameter."""
```

```
        Frame.__init__(self)
        self.n_disks = n_disks
        self.bars = [ [], [], [] ]

        self.createGUI()

    #------------

    def createGUI(self):
        """Initializes the GUI, creating the floor, pegs and
           disks."""

        # Create main canvas
        self.master.title('Torres de Hanói')
        self.canvas = Canvas(self, width=Hanoi.width, height=
           Hanoi.height)
        self.canvas.pack()
        self.pack()

        # Object sizes
        floor_size = Hanoi.width-2*Hanoi.dx
        self.disk_height = (Hanoi.height - 4*Hanoi.sep) / self
           .n_disks
        self.disk_width = (floor_size - 4*Hanoi.dx) / 3

        # Shorthand notation
        disk_w = self.disk_width
        disk_h = self.disk_height
        width = Hanoi.width
        height = Hanoi.height
        sep = Hanoi.sep
        dx = Hanoi.dx

        # Draw floor
        self.canvas.create_rectangle(dx, height-sep, width-dx,
            height,
                                     fill='brown', outline='
                                        brown')

        # draw bars
```

```
        bar_pos = [ 2*dx + disk_w/2 + (disk_w+dx)*i for i in
           range(0,3) ]
        for x in bar_pos:
            self.canvas.create_rectangle(x-dx, sep, x+dx,
               height-sep-1, fill='black')

        # create disks
        for i in range(self.n_disks):
            disk = self.canvas.create_rectangle(2*dx + 4*dx*i,
                height-sep-1-disk_h*i,
                                                2*dx + disk_w
                                                   - 4*dx*i,
                                                   height-sep
                                                   -1-disk_h*(
                                                   i+1),
                                                fill='blue')
            self.bars[0].append(disk)

        self.update()

    #---------------

    def moveDisk(self, origin, dest):
        """Moves a disk between an origin peg and a
           destination peg. Pegs are
           numbered from 0 to 2."""

        # Move the disk in terms of data structures
        disk = self.bars[origin].pop()
        self.bars[dest].append(disk)

        # Move the disk on the GUI
        dx = (dest-origin) * (self.disk_width + Hanoi.dx)
        dy = ( len(self.bars[origin])-len(self.bars[dest])+1 )
            * self.disk_height

        self.canvas.move(disk, dx, dy)
        self.canvas.update()
        time.sleep(Hanoi.delay)

    def run(self):
```

```
        """Main method of the GUI, which starts the simulation
           ."""

        hanoi(self.n_disks, 0, 2, 1, self.moveDisk)

#--------------------------

if __name__ == "__main__":
    #hanoi(4,'A','B','C')
    gui = Hanoi(4)
    time.sleep(1)
    gui.run()
```

Exercício 14.22 MD

Solução

```
from random import *
from tkinter import *


class CA_1D:

  def __init__(self, rule,generations=75,grid_size=150):
    self._result = None
    self._decompose(rule)
    self._generations = generations
    self._grid_size = grid_size

  def run(self):
    g0 = [0]*(self._grid_size//2) + [1] + [0]*(self._grid_size
       //2)
    result = [g0]
    last = g0
    for i in range(self._generations):
      left_elem = self._apply_rule(last[self._grid_size], last
         [0], last[1])
      right_elem = self._apply_rule(last[self._grid_size - 1],
          last[self._grid_size], last[0])
      other_elem = [self._apply_rule(last[i-1], last[i], last[
         i+1]) for i in range(1, self._grid_size)]
```

```
      new= [left_elem] + other_elem + [right_elem]

      result.append(new)
      last = new

    self._result = result
    return result

  def _apply_rule(self, left, center, right):
    return self._rule_template[ left*4 + center*2 + right ]


  def _decompose(self, n):
    self._rule_template = []
    for i in range(8):
      r = n%2
      n = n//2
      self._rule_template.append(r)

class Display:
  def __init__(self,autocel):
    self._result = autocel._result
    self._grid_size = autocel._grid_size
    self._generations = autocel._generations


  def draw(self):
    if self._result == None:
      print("Ainda nada para ver...")
      return
    d = 2
    width  = d*(self._grid_size + 1)
    height = d*(self._generations + 1)
    canvas = Canvas(width=width, height=height, bg='white')
    canvas.master.title('Simulador de um Autómato Celular')
    canvas.pack(side=LEFT,expand=YES,fill=BOTH)
    x = 2
    y = 2
    for line in self._result:
      for block in line:
```

```
        if block==0:
          canvas.create_rectangle(x, y, x+d, y+d, fill='white'
              , width=1)
        else:
          canvas.create_rectangle(x, y, x+d, y+d, fill='black'
              , width=1)
        x =  x + d
      y= y + d
      x = 2


class MyApp():
  def __init__(self,rule,generations,grid_size):
    self._ca = CA_1D(rule,generations,grid_size)
    self._ca.run()
    self._d = Display(self._ca)
    self._d.draw()


if __name__ == '__main__':
  # Número da Regra, Gerações, Tamanho da Grelha
  app1=MyApp(30,75,150)
  #app2=MyApp(31,75,150)
  mainloop()
```

Exercício 14.23 MD

Solução

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  7 14:34:58 2015

@author: ernestojfcosta
"""

from tkinter import *
from math import *

# Sistema L
class SistemaL:
```

```
  """Um sistema L tem um axioma, regras e uma forma sentencial
      ."""
  # construtor
  def __init__(self,axioma,regras,frase=None):
    self._axioma=axioma
    self._regras = regras
    if frase:
      self._frase=frase
    else:
      self._frase=axioma

  # Selectores
  def obtem_axioma(self):
    return self._axioma

  def obtem_regras(self):
    return self._regras

  def obtem_frase(self):
    return self._frase

  # Modificadores
  def reescreve(self):
    self._frase= ''.join([self._regras.get(ch,ch) for ch in
       self._frase])

  # Auxiliares
  def show(self):
    print('Axioma: ', self._axioma)
    print('Regras: ', self._regras)
    print('Frase: ', self._frase)



class Tartaruga:

  def __init__(self,posx=300,posy=300,heading=-90,estado='up',
     tamanho=5,angulo=90):
    self._posx = posx
    self._posy = posy
    self._heading = heading
```

```
    self._estado = estado
    self._tamanho = tamanho
    self._angulo = angulo

    self._pilha = []

    self._dicio={'F':'self.tart_down();self.move(ecran);self.
        tart_up()',
    'B':'pass','f':'self.tart_up();self.move(ecran)',
    '+':'self.rodad()','-':'self.rodae()',
    '[':'self.guarda_estado()',']':'self.restaura_estado()' ,
    'D':'self.tart_down();self.move(ecran);self.rodad();self.
        move(ecran);self.tart_up()',
    'E':'self.tart_down();self.move(ecran);self.rodae();self.
        move(ecran);self.tart_up()'}

# Comandos da tartaruga
# Movimento
def move(self,ecran):
    posxx = self._posx + self._tamanho * cos(radians(self.
        _heading))
    posyy = self._posy + self._tamanho * sin(radians(self.
        _heading))
    if self._estado == 'up':
      self._posx = posxx
      self._posy = posyy
    else:
      ecran.create_line(self._posx,self._posy,posxx,posyy)
      self._posx = posxx
      self._posy = posyy

# roda a tartaruga
def rodad(self):
    self._heading += self._angulo

def rodae(self):
    self._heading -= self._angulo

# controla caneta
def tart_up(self):
    self._estado='up'
```

```
  def tart_down(self):
    self._estado='down'

  # Gestao do estado da pilha
  def guarda_estado(self):
    posicx = self._posx
    posicy = self._posy
    heading =self._heading
    self._pilha=[[posicx,posicy,heading]] + self._pilha

  def restaura_estado(self):
    posicx,posicy,heading = self._pilha[0]
    self._posx = posicx
    self._posy = posicy
    self._heading = heading
    self._pilha = self._pilha[1:]

  def obtem_codigo(self,simbolo):
    return self._dicio[simbolo]

  def executa_codigo(self,codigo,ecran):
    exec(codigo)


class Ecran(Frame):

  def __init__(self):
    super().__init__()
    self.pack(expand=1,fill=BOTH)
    self.master.title('Simulador de Sistemas de Lindenmayer')

    self._ax = Label(self, text='Axioma')
    self._ax.pack()
    self._var_ax=StringVar()
    self._ax_ent = Entry(self,textvariable=self._var_ax)
    self._ax_ent.pack()

    self._regras = Label(self, text='Regras')
    self._regras.pack()
    self._var_regras=StringVar()
```

```
    self._regras_ent = Entry(self,textvariable=self.
       _var_regras)
    self._regras_ent.pack()

    self._frase = Label(self, text='Frase')
    self._frase.pack()
    self._var_frase=StringVar()
    self._frase_ent = Entry(self,textvariable=self._var_frase,
        width=50)
    self._frase_ent.pack()


    self._tela = Canvas(self,width=600,height=600,bg='light
       yellow')
    self._tela.pack(expand=YES,fill=BOTH)

  def mostra_axioma(self,axioma):
    self._var_ax.set(axioma)

  def mostra_regras(self, regras):
    self._var_regras.set(regras)

  def mostra_frase(self, frase):
    self._var_frase.set(frase)

  def create_line(self, x1,y1,x2,y2):
    self._tela.create_line(x1,y1,x2,y2)


class Aplicacao:

  def __init__(self,axioma,regras,frase, passos):
    self._sl = SistemaL(axioma,regras,frase)
    self._ecran = Ecran()
    self._tartaruga = Tartaruga()
    self._passos = passos

    self._ecran.mostra_axioma(self._sl.obtem_axioma())
    self._ecran.mostra_regras(self._sl.obtem_regras())

  def run(self):
```

```
    # Passo 1: Executa o sistema L
    for i in range(self._passos):
      self._sl.reescreve()
    self._frase = self._sl.obtem_frase()
    self._ecran.mostra_frase(self._sl.obtem_frase())

    for simbolo in self._frase:
      # Passo 2: traduz comando em codigo
      self._codigo = self._tartaruga.obtem_codigo(simbolo)
      # Passo 3: Executa codigo
      self._tartaruga.executa_codigo(self._codigo,self._ecran)
    mainloop()

if __name__ == '__main__':
     axioma ='E'
     regras = {'E':'E+D+','D':'-E-D'}
     frase = None
     passos = 10
     aplica = Aplicacao(axioma, regras,frase, passos)
     aplica.run()
```