UNIVERSIDADE DE ÉVORA

Escola de Ciências e Tecnologias

Engenharia Informática

Programação I

2017/2018

Trabalho prático 1

- Pigs and Bulls -

Ana Ferro – 39872

Docente: Teresa Gonçalves

Índice

Introdução

O presente trabalho foi desenvolvido no âmbito da disciplina de Programação I e consiste no desenvolvimento de uma aplicação para jogar *Pigs and Bulls* com o computador.

O objetivo deste jogo é tentar adivinhar um código de 4 algarismos diferentes gerado pelo computador, para tal o jogador vai fazendo tentativas até conseguir adivinhar. Em cada tentativa é indicado o número de acertos. Dependente do tipo de acerto é indicado o número de touros ou/e o número de porcos. Os touros correspondem aos algarismos que estão nas posições certas e os porcos correspondem aos algarismos certos que estão em posições erradas. O jogo acaba quando o jogador acertar o código.

Este trabalho tem como objetivo implementar os conhecimentos adquiridos nas aulas, para tal o programa foi desenvolvido em *Python 3.6* .

Com vista a melhorar a execução deste trabalho, este foi dividido em duas fases:

- Fase 1- Esta fase consiste em compreender o problema que é apresentado, dividi-lo em problemas mais simples, encontrar as soluções para estes e, assim, chegar à solução principal.

- Fase 2 – Esta fase consiste em implementar e testar o programa obtido na fase anterior

Pigs and Bulls

Fase 1

- Problema:

  Como anteriormente foi descrito na introdução, o computador gera um código de 4 algarismos entre 0 e 9, sem repetição. O jogador vai tentar adivinhar esse código. Em cada tentativa, o programa indica o nº de touros e/ou o nº de porcos. Quando o jogador acertar, o programa devolve o nº de tentativas, tal como a sequência de tentativas e respetivos acertos.

  Input: código de 4 algarismos introduzido pelo jogador
  Output: - Se acertar devolve o nº de tentativas, tal como a sequência de tentativas e respetivos acertos.
  - Se errar devolve o nº de touros e/ou o nº de porcos

- Divisão do problema:

  Depois de conhecido o problema e da compreensão do que é pedido dividiu-se o problema geral em 6 problemas mais simples, apresentando-os em forma de pergunta e á qual se deu resposta. Esses problemas consistem em:

  1.Como gerar um código aleatório sem repetição?
  2.Como fazer a contagem do nº de touros e/ou do nº de porcos?
  3.Como fazer com que o programa continue a pedir um novo código até que o jogador acerte?
  4.Como apresentar o nº total de tentativas tal como a sequência de tentativas e respetivos acertos?
  5. Como dar a conhecer as regras do jogo ao jogador?
  6. Como excluir as tentativas inválidas?

- Respetivas respostas:

  1.Como gerar um código aleatório sem repetição?

  Para dar resposta a este problema foi necessário a utilização da função *random()* do módulo *random* que, por si só, gera um número aleatório no intervalo [0,1[. Com recurso a essa função criou-se uma outra função, a função *código_aleatorio().*

  Esta nova função cria um número de 4 algarismos, cada um diferente dos restantes. Para assegurar que não haja repetição de algarismos recorreu-se á utilização de ciclos *while*. Quando o computador gerar com sucesso um código válido, retorna esse código e "guarda-o" para posteriormente ser chamado pelas restantes funções.

2.Como fazer a contagem do nº de touros e/ou do nº de porcos?

Foi criada a função *jogo,* que tem como argumentos o código gerado pelo computador utilizando a função anterior e o código que o jogador posteriormente irá introduzir.

Esta função tem como objetivo analisar o código introduzido pelo jogador e contar o nº de touros e o nº de porcos. Basicamente, esta função vai comparar o código introduzido e o código gerado, para tal recorreu-se a ciclos *for* e às funções *range()* e *len().*

3. Como fazer com que o programa continue a pedir um novo código até que o jogador acerte?

Para que tal seja possível, foi necessário implementar um ciclo *while*. Dependendo do código inserido pelo jogador, o ciclo comporta-se de diferentes modos (existência de estruturas condicionais no ciclo *while*). Se o jogador acertar o código, aparecerá uma mensagem de que o jogador acertou e o jogo chega ao fim. Se o jogador errar, a função *jogo* será chamada e irá devolver o nº de touros e/ou de porcos correspondente, consequentemente, é pedido um novo input ao jogador.

4. Como apresentar o nº total de tentativas tal como a sequência de tentativas e respetivos acertos?

A resposta a este problema está diretamente relacionada com a questão anterior, daí utilizar o mesmo ciclo *while*. Ainda fora do ciclo foi necessário criar uma variável igualada a zero que vai funcionar como um contador e outras duas variáveis vazias. Este contador irá contar as vezes que o programa vai percorrer o ciclo, parando quando o jogador acertar no código. E as variáveis vazias serão utilizadas para guardar as tentativas do jogador e respetivos acertos.

Quando finalmente acertar, será devolvido o nº total de tentativas tal como a sequência de tentativas e respetivos acertos.

5. Como dar a conhecer as regras do jogo ao jogador?

Foi criada a função *regras_do_jogo*, que funciona como uma pequena introdução ao jogo, dando a conhecer as suas regras a quem porventura o jogar.

Foram, então, criadas várias strings que no final serão devolvidas pela função através da sua concatenação.

6. Como excluir as tentativas inválidas?

Para diminuir possíveis erros foi adicionada a função *validade_code*, que vai avaliar a validade do código introduzido pelo jogador. Ou seja, vai confirmar se o input cumpre os critérios do jogo:

- Se, de facto, o código possui 4 algarismos;
- Se possui apenas valores numéricos;
- E se não possui nenhum algarismo repetido.

Se tais critérios não forem cumpridos, a função devolve a mensagem "Código inválido" e, consequentemente, será pedido ao jogador que introduza um novo código.

Fase 2

Esta fase consiste na codificação e consequente teste das funções desenhadas na fase 1, tanto individualmente como em conjunto.

```
import random

def codigo_aleatorio():
    a=str(int(random.random()*10))
    b=str(int(random.random()*10))
    c=str(int(random.random()*10))
    d=str(int(random.random()*10))
    while a==b or a==c or a==d:
        a=str(int(random.random()*10))
    while b==c or b==d or b==a:
        b=str(int(random.random()*10))
    while c==d or c==a or c==b:
        c=str(int(random.random()*10))
    while d==a or d==b or d==c:
        d=str(int(random.random()*10))
    code=a+b+c+d
    return code
```

- Função *código_aleatorio:*

  Criação de 4 variáveis, cada uma correspondente a um diferente algarismo do código. Uma vez que a função *random()* do módulo *random()* devolve um *float* no intervalo de [0,1[foi necessário multiplicar por 10 para expandir o intervalo, [0.0, 10.0[. De seguida, foi necessário recorrer a uma conversão explícita de *float* para *int*, pois é pretendido o computador criar um código com algarismos entre 0 e 9, ou seja, no intervalo de [0,10[.

  Foram implementados 4 ciclos *while*, com o intuito de gerar um código sem algarismos repetidos. No primeiro *while* certifica-se se a primeira variável(*a*) é igual às restantes variáveis (*b,c* e *d*), se isso se confirmar é gerada uma nova

variável *a*, se não se confirmar prossegue-se para o seguinte *while*. Com a mesma lógica que o primeiro *while*, o segundo *while* certifica-se se a segunda variável(*b*) é igual às restantes variáveis (*a,c* e *d*) e tal como o *while* anterior, se se confirmar é gerada uma nova variável *b*, se se não confirmar prossegue-se para o seguinte *while*. Este raciocínio continua no terceiro *while* e no quarto *while*.

Recorreu-se também a uma conversão explícita de *int* para *str*, para ser possível fazer a concatenação das variáveis sem perderem a sua individualidade enquanto algarismo.

A função vai retornar um código de 4 algarismos diferentes em formato de *str*.

Esta função foi uma das mais desafiantes. Numa primeira fase recorreu-se a estruturas condicionais, no entanto, não excluía a repetição de algarismos e por vezes nem gerava um número apresentando invés a mensagem "None".

```
def jogo(aleatorio,code):
    bulls=0
    for i in range (len(code)):
        if code[i]==aleatorio[i]:
            bulls=bulls+1
    pigs=0
    for i in range(len(code)):
        for x in range(len(aleatorio)):
            if code[i]==aleatorio[x]:
                pigs=pigs+1
    pigs=pigs-bulls

    if bulls==0:
        return(str(pigs)+'P'+'')
    elif pigs==0:
        return (str(bulls)+'T'+'')
    else:
        return(str(bulls)+'T'+' '+str(pigs)+'P'+' ')
```

- Função *jogo*:

  Esta função tem como argumentos o código gerado pelo computador(aleatorio) e o código introduzido pelo jogador(code) e é pretendido devolver o nº de touros e/ou o nº de porcos.

  Esta função possui 3 partes lógicas. A primeira é correspondente à contagem do nº de touros, a segunda parte é correspondente à contagem do nº de porcos e a terceira corresponde à mensagem de retorno adequada.

  1ºParte

  Criou-se uma variável denominada de *bulls* e igualou-se a zero. Esta variável vai funcionar como um contador. De seguida, criou-se um ciclo de repetição recorrendo a um ciclo *for* e à função *range()*. Este ciclo percorre e analisa todo o comprimento do código introduzido pelo jogador índice a índice e

vai compará-lo com o código gerado pelo computador. Quando a estrutura condicional se confirmar ou seja, quando, para os mesmos índices, se encontrar algarismos iguais, soma-se uma unidade à variável. A variável vai sendo alterada quando se confirma a condição e termina quando se acabou de analisar o código.

2ºParte

Tal como na contagem dos touros, foi também criada uma variável igualada a zero cujo nome é *pigs*. Para descobrir os algarismos certos que estavam em posições erradas recorreu-se a um ciclo for dentro de um ciclo for. Este loop vai percorrer o comprimento do código inserido e o comprimento do código gerado. Desta maneira vai-se certificar se qualquer algarismo do código introduzido, independentemente da posição onde se encontra, é igual aos algarismos constituintes do código gerado. Quando se encontra uma igualdade de algarismos, soma-se uma unidade à variável *pigs*. Quando se acabar de percorrer ambos os códigos, a variável *pigs* possui um certo valor.

Para chegar a um valor final é necessário subtrair o nº de touros ao nº de porcos, uma vez que o nº de algarismos certos em posições corretas é também contabilizado.

3ºParte

Por último foi criado uma estrutura condicional. Por mera questão estética, decidiu-se apresentar três possível situações: quando houvesse apenas touros, quando houvesse apenas porcos ou ambos. Dependente da condição assim era o seu retorno. A função retorna os valores das variáveis em formato de *str*, para tal foi necessário fazer uma conversão explícita de *int* para *str*.

O principal problema da execução desta função foi mesmo quando se pedia o retorno do nº de porcos e este incluir também no seu valor o nº de touros. A resolução deste problema foi simples, bastando acrescentar uma linha de código ao programa e que consistia em subtrair o nº de touros, calculados previamente, ao nº de porcos.

```
def regras_do_jogo():
    a='  '*5+'PIGS AND BULLS\n'
    b='O computador gera um código de 4 algarismos entre 0 e 9, sem repetição.\n'
    c='O objetivo deste jogo é adivinhar esse código, para isso o jogador vai fazendo várias tentativas até acertar.\n'
    d='Em cada tentativa o programa devolve o nº de acertos.\n'
    e='Se os algarismos estão nas posições certas, o programa indica o nº de touros(T).\n'
    f='E se os algarismos certos estão em posições diferentes, o programa indica o nº de porcos(P).\n'
    g='Por exemplo:\n'
    h='Se o código for 4271 e a tentativa for 1234, a resposta deve ser 1T, 2P (o touro é "2", os porcos são "4" e "1").\n'
    i='Vamos jogar?'
    return a+b+c+d+e+f+g+h+i
```

- Função *regra_do_jogo*:

  Apesar de não ser pedido, foi criada esta função com intuito de dar a conhecer as regras do jogo ao jogador. Para tal foram criadas diversas strings. O

retorno desta função é a concatenação de todas essas strings. Utilizou-se também o caractere \n para que, quando o código fosse executado, cada string ocupasse uma linha diferente invés de se apresentar como texto corrido.

```
def validade_code(my_guess):
    alfabeto=['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z']
    for i in range(len(alfabeto)):
        for x in range(len(my_guess)):
            if my_guess[x]==alfabeto[i]:
                return'Código inválido'

    if len(my_guess)!=4:
        return 'Código inválido'
    elif my_guess[0]==my_guess[1] or my_guess[0]==my_guess[2] or my_guess[0]==my_guess[3] or my_guess[1]==my_guess[2] or my_guess[1]==my_guess[3] or my_guess[2]==my_guess[3]:
        return 'Código inválido'
```

- Função *validade_code*:

  Esta função foi usada para confirmar a validade do código introduzido pelo jogador e para restringir possíveis erros relativamente à contagem de touros e porcos, uma vez que é diretamente influenciado pelo input do jogador. Para que o código seja válido tem que cumprir 3 critérios.

  1ºCritério
  O código introduzido não pode possuir valores não numéricos. Para o confirmar foi criada uma lista com todas as letras do alfabeto. De seguida criou-se uma estrutura de repetição com um ciclo for dentro de um ciclo for. Desta maneira, vai-se comparar todo o comprimento da lista com o comprimento do código. Se se encontrar uma correspondência entre a lista e o código a função retorna a mensagem “Código inválido”.

  2ºCritério
  O código introduzido não pode ter menos ou mais que 4 algarismos. Para confirmar este critério criou-se uma estrutura condicional. Se o comprimento do código introduzido é diferente de 4 a função retorna a mensagem “Código inválido”. Para calcular o comprimento do código introduzido utilizou-se a função *len().*

  3ºCritério
  O código introduzido não pode possuir números repetidos. Tal como no critério anterior foi necessária uma estrutura condicional. Nela estão introduzidas todas as combinações possíveis de igualdade de números. Ao utilizar o operador booleano *or*, basta uma das expressões ser verdadeira para que a função retorne a mensagem “Código inválido”.

```
print(regras_do_jogo())
code=codigo_aleatorio()
my_guess=0
tentativas=0
a=''
b=''
while my_guess!=code:
    my_guess=input('\nCódigo?')
    if validade_code(my_guess):
        print (validade_code(my_guess))
        continue

    tentativas=tentativas+1
    a=''+'T'+str(tentativas)+':'+ my_guess + ','+ jogo(code, my_guess)+ '\n'
    b=b+a

    if my_guess == code:
        print('Acertou!!!')
        print('\nAs tuas tentativas foram: \n'+b)
    else:
        print(jogo(code,my_guess))
```

Este excerto corresponde à parte final do programa onde são invocadas as funções anteriores e onde é feita a contagem das tentativas.

Em primeiro lugar invoca-se a função *regras_do_jogo*, que será a primeira mensagem a aparecer quando o programa for executado. De seguida foi criado 4 variáveis: a variável *my_guess* igualada a zero, a variável *tentativas* também igualada a zero e as variáveis vazias *a* e *b*.

Para que o programa peça continuamente ao jogador que introduza um código foi implementado um ciclo *while*. Dentro deste ciclo, em primeiro lugar, vai ser chamada a função *validade_code*, se o código não é válido é devolvida uma mensagem de erro e será pedido de novo que seja introduzido outro código, se o código for válido continua-se a executar o que está dentro do ciclo *while*.

De seguida, encontra-se o contador das tentativas, ou seja, cada vez que esta linha de código é executada à variável denominada de *tentativas* é adicionada uma unidade. No final, quando o jogador acertar o código esta variável será equivalente ao nº total de tentativas (os códigos inválidos não são contabilizados).

Na linha seguinte, a variável *a* vai guardar a informação sobre o nº da tentativa, a sequência da tentativa e os respetivos acertos. Enquanto que, a variável *b* vai guardar todas as diferentes informações da variável *a* e que será mostrada posteriormente.

Ainda dentro do ciclo *while*, recorrendo a estruturas condicionais comparou-se o código inserido pelo jogador e o código gerado pelo computador. Se o código inserido for exatamente igual ao código gerado é devolvido uma mensagem de sucesso e as respetivas tentativas guardadas na variável *b*. Se o código for diferente, invoca-se a função *jogo* que devolve o nº de touros e/ou porcos.

O ciclo *while* continua a ser executado até o jogador acertar no código.

Conclusão

O desenvolvimento de uma aplicação para jogar *Pigs and Bulls* foi rico em aprendizagem. Não só consolidou os saberes adquiridos em sala de aula, bem como foi introduzido um novo módulo matemática do *Python:* o *random.*

As dificuldades foram algumas, mas depois de ultrapassadas foi possível aperfeiçoar o trabalho e tornar o código mais eficiente.

Bibliografia

- Protocolo

- http://excript.com/python/for-in-range-python.html

- http://excript.com/python/while-else-python.html

- https://cscircles.cemc.uwaterloo.ca/9-else-and-or-not/~

- https://code.tutsplus.com/pt/tutorials/mathematical-modules-in-python-random--cms-27738

- https://docs.python.org/2/tutorial/controlflow.html