Curso de Computação 1
Introdução à Programação em Python

AULA 7

Estrutura de repetição com teste de parada: While

Estamos escrevendo nossos códigos em Python no estilo modular de programação, onde definimos módulos, representados por funções em Python, para realizar tarefas específicas, e a articulação destas funções leva à realização de tarefas maiores.
Para escrever nossas funções, usamos os recursos que o Python nos oferece: tipos de dados, operações, funções já prontas, variáveis, atribuição e estruturas como a de seleção (IF). Nesta aula, veremos mais uma estrutura de programação: a estrutura de repetição (ou loop).

1. Introdução a estruturas de repetição

Estruturas de repetição, como o nome já evidencia, são estruturas que fazem repetições de tarefas de acordo com algum critério definido. Esse tipo de situação é bastante comum. Por exemplo, quando se deseja limpar uma superfície deve-se passar o pano com o material limpante até que a superfície esteja limpa. Ou seja, *enquanto* a superfície estiver suja, deve-se repetir a tarefa de passar o pano úmido.

No primeiro vídeo, falaremos das principais características das estruturas de repetição e de seu propósito na programação.

Vídeo: Introdução às estruturas de repetição

Agora vamos ver um exemplo de problema que só conseguimos tratar quando fazemos uso de repetições em nosso algoritmo. No laboratório da aula 4 na Machine Teaching vimos o problema da filtragem, com o seguinte enunciado, que especificava que a tupla de entrada tinha que ter exatamente 4 elementos:

> ***Vale a pena ver de novo:*** Faça uma função chamada **filtra_pares** que receba uma tupla com quatro elementos inteiros como argumento, e retorne uma nova tupla contendo apenas os elementos pares da tupla original, na mesma ordem em que se encontravam. Esse tipo de operação onde se selecionam elementos de um conjunto inicial que satisfazem uma determinada propriedade é bastante comum em computação, e se chama **filtragem**.

Se você não lembra do problema ou da solução ou se tem interesse em ouvir uma discussão sobre possíveis soluções para o problema, pode assistir esse vídeo.

Vídeo: Problema da filtragem com entrada de tamanho 4

E nesse próximo vídeo apresentamos a generalização do problema e da solução para tratar entradas com pelo menos um elemento, ou seja, com qualquer quantidade de elementos que não seja zero. Uma solução para este problema mais geral precisa fazer uso de estruturas de repetição.

Vídeo: Problema da filtragem com entrada de tamanho livre

2. Comando de repetição com teste de parada: *while*

Agora que você já teve um primeiro contato com os tipos de problemas que queremos resolver e como estruturas de repetição são necessárias para resolvê-los, você verá, no próximo vídeo, quais são as semelhanças e diferenças entre o comando **while** e estrutura condicional **if** que você já conhece.

Vídeo: Comando while

Vamos fazer agora o teste de mesa de um código contendo um comando while. No teste de mesa fazemos uma simulação de execução do código, o que é uma forma poderosa para verificar o correto entendimento do funcionamento dos comandos utilizados. Já fizemos vários testes de mesa antes, mas agora, o fato de estarmos lidando com variáveis que a cada repetição podem mudar de valor, é preciso tomar um pouco mais de cuidado com como registrar a informação de maneira clara. Vamos ver isso no vídeo.

Vídeo: Simulação de execução de código contendo um comando while

Relembrando: para acessar a ferramenta Python tutor, mencionada no vídeo, use o link http://www.pythontutor.com/

Atividade: Vamos agora treinar um pouco com exercícios de fixação. Responda as perguntas da atividade “While: primeiros passos”. Fique atento ao prazo de entrega dessa atividade!

Curso de Computação 1
Introdução à Programação em Python

3. Contadores e acumuladores

Vamos retomar o exemplo do exercício “filtra pares”. Você reparou que, para resolver esse problema no caso de uma tupla de qualquer tamanho, usamos duas variáveis que eram atualizadas a cada repetição? Uma delas para guardar o índice do elemento a ser verificado, e outra, para construir gradativamente a tupla de resposta. Essas duas estratégias de atualização de variáveis associadas à estrutura de repetição são bastante comuns, e recebem nomes especiais: **contadores** e **acumuladores**, respectivamente.

Contadores e acumuladores são conceitos teóricos. Nas linguagens de programação não temos elementos sintáticos que correspondam a eles. Para o computador tanto contadores quanto acumuladores são simplesmente variáveis. Estes conceitos fazem no entanto diferença na idealização da construção do seu código.

Um contador é uma variável criada normalmente com o intuito de contar quantas vezes alguma ação foi executada. Contadores são muitas vezes usados na condição de parada do while. Se eu quero executar uma determinada ação n vezes, posso criar um contador que me diz quando esse n foi atingido. No exemplo filtra_pares, esse n era o tamanho da tupla de entrada, a ação que estávamos contando era a verificação de cada elemento da tupla, e o contador foi implementado pela variável *próximo*. Verifique estes elementos no código da função:

```
def filtra_pares_v3(t):
    ''' funcao que dada uma tupla não vazia de inteiros, retorna uma tupla com os
    inteiros pares da tupla original, mantida a ordem.
    tuple --> tuple'''
    pares = ()
    proximo = 0
    while proximo <len(t):
        if t[proximo]%2 == 0:
            pares = pares + (t[proximo],)
        proximo = proximo + 1
    return pares
```

A idéia por trás da noção de acumulador, por outro lado, é a construção passo a passo da resposta de uma função ou de alguma informação que vai ser usada posteriormente para a construção dessa resposta. Era o caso da variável *pares*, no problema filtra_pares. A cada iteração do while, se um novo número par foi identificado, ele foi incluído na resposta.

O contador é bem representado por uma variável do tipo inteiro. Já o tipo de dado da variável acumuladora dependerá do que se deseja "acumular". No do problema de filtragem de pares, nosso desejo era acumular elementos pares em uma tupla, logo o acumulador também era uma tupla.

ATENÇÃO! Em todos os casos, é essencial inicializar CORRETAMENTE as variáveis contadoras e acumuladoras antes do início da execução do laço! Leve em conta o tipo de dado e a operação de acumulação sendo feita em cada caso.

Veja outras possibilidades nos exemplos abaixo:

Exemplo 1: *Qual é a última vogal de uma palavra dada?*

```
def ultima_vogal(palavra):
    '''retorna a ultima vogal da palavra, ou vazio
    se a palavra não tiver vogais
    str->str'''
    i=0
    vogal=''
    while i<len(palavra):
        if palavra[i] in 'AEIOUaeiou':
            vogal=palavra[i]
        i=i+1
    return vogal
```

Neste exemplo, a variável i está fazendo o papel de um contador. Não temos nenhum acumulador sendo usado, pois este conceito não foi necessário para a construção da resposta, que é diretamente a última vogal encontrada.

Exemplo 2: *Qual o número de vogais em uma string.*

```
def quantidade_vogais (texto):
    ''' devolve a quantidade de vogais presentes no texto.
    str --> int'''
    qtd_vogais = 0
    i = 0
    while i < len(texto):
        if texto[i] in 'AEIOUaeiou':
            qtd_vogais = qtd_vogais + 1
        i = i + 1
    return qtd_vogais
```

Neste exemplo, temos tanto um contador, implementado pela variável i, quanto um acumulador, implementado por qtd_vogais. A cada iteração do laço, no momento da avaliação da condição de parada, i indica quantas posições do texto já foram verificadas. Ao final da execução do laço, a variável qtd_vogais registra quantas vogais foram identificadas no texto.

Exemplo 3: *Qual a sequência de vogais de um texto?*

```
def todasasvogais(texto):
    '''retorna uma string com as vogais que apareceram em um texto,
    na mesma sequencia que apareceram
    str->str'''
    i=0
    vogais=''
    while i<len(texto):
        if texto[i] in 'AEIOUaeiou':
            vogais=vogais+texto[i]
        i=i+1
    return vogais
```

Neste exemplo temos um contador e um acumulador. Você consegue identificá-los? O acumulador é do tipo string.

Atividade: Vamos agora treinar um pouco com exercícios de fixação. Responda as perguntas da atividade “Contadores e acumuladores”. Fique atento ao prazo de entrega dessa atividade!

Veremos agora outros exemplos de código com o comando **while**. Você também verá o uso da função **randint** para manipular valores randômicos (aleatórios).

Vídeo: Mais exemplos de uso do comando while

Observação: O random é um módulo do Python dedicado à geração de números aleatórios. Ele tem várias funções interessantes, além da função randint abordada no vídeo. Vale a pena olhar mais sobre este módulo (usando o help no IDLE ou olhando a documentação na internet), pode ser útil em outras disciplinas do seu curso, não apenas Computação 1.

4. Loop infinito e retorno precoce

O próximo vídeo mostrará os erros mais comuns que podem ocorrer ao se utilizar o comando **while**. É importante conhecer esses erros, saber os seus resultados e entender como evitá-los.

Vídeo: Erros comuns no uso do comando while

Curso de Computação 1
Introdução à Programação em Python

No arquivo de códigos desta aula 7, você encontrará versões corretas e erradas de algumas das funções vistas ao longo deste roteiro de estudos. Execute-as no Python tutor para fixar seus conhecimentos sobre os conceitos desta aula e aprender a identificar os erros comuns no uso do comando while. Pense bem nos testes que você vai executar para identificar os erros das versões defeituosas.

Observações importantes:

- Quando pedimos para o computador executar um laço infinito, ou seja, um laço cuja condição de parada nunca é verdadeira, parece que o interpretador está travado pois não aparece resposta nenhuma. Neste caso, você pode teclar a combinação de teclas Control+C para interromper forçadamente a execução.
- Em alguns casos, no entanto, apesar do laço ser infinito em teoria, o fato dos dados que estamos usando no computador serem finitos, pode levar a uma situação onde a condição de parada eventualmente é satisfeita, com a obtenção de uma resposta que, em geral, não faz muito sentido. É o caso do problema das populações que crescem. Temos a população A (popA) e a população B (popB) e suas respectivas taxas de crescimento, e queremos saber quando popA alcança popB. Caso popA seja menor do que popB e sua taxa de crescimento seja também menor, a condição de parada (popA >= popB) nunca deveria ser verdade. Mas na realidade o tipo de dados float é finito, e quando as populações obtidas ultrapassarem o limite de valores representáveis pelo tipo float, essas variáveis receberão uma sequência de zeros e uns que é interpretada como **infinito**, por convenção. Quando ambas chegarem a esse valor, as populações serão consideradas iguais, levando à parada do laço e o retorno de algum valor que não representa o que o programador desejava. Fique atento!
- Algumas ferramentas (não é o caso do IDLE) incluem o conceito de “time-out”, ou seja, ela executa por um determinado tempo e eventualmente “desiste” de continuar, encerrando a execução do código. O que será devolvido como resposta vai depender da ferramenta, nesse caso.

Atividade: Vamos agora treinar um pouco com exercícios de fixação. Responda as perguntas da atividade “Explorando o while”. Fique atento ao prazo de entrega dessa atividade!

Prática em Programação

Após concluir as etapas anteriores deste roteiro, faça as atividades práticas desta aula, disponíveis no Google Classroom da turma.

Até a próxima aula!