Apostila de Introdução à Programação
Prof. Eduardo Corrêa

Capítulo VI –Programação Python – Estruturas de Repetição: while e for

Sumário

VI. Estruturas de Repetição: while e for

O capítulo anterior cobriu o desvio condicional, que é um dos tipos de instrução de controle utilizadas em programas. Este capítulo introduz um outro tipo de instrução de controle igualmente importante: a **repetição**.

VI.1 Repetição

Uma estrutura de repetição permite com que um conjunto de instruções possa ser executado diversas vezes **até que uma condição seja satisfeita**. Na Linguagem Python existem dois comandos de repetição: ***while*** e ***for***. Eles serão objeto de estudo desse capítulo.

Por que os comandos de repetição existem? Um primeiro motivo, é o fato de que diversos problemas práticos têm a sua resolução imensamente facilitada com o uso dos comandos de repetição. Considere o seguinte exemplo:

EXEMPLO VI.1. Elabore um programa que receba a nota final de 50 alunos de uma turma de Física I como entrada e que calcule e imprima a nota média da turma.

Para resolver este problema sem o uso de comandos de repetição, seria necessário, implementar 50 comandos **readln** (para ler a nota dos 50 alunos) e mais 50 linhas para atualizar a soma das notas, conforme apresenta a Figura 1.

```
#Soma 50 números sem repetição (não faça isso!!!)
soma = 0
print('Digite 50 notas: ')
nota = float(input("nota1 : ")); soma = soma + nota
nota = float(input("nota2 : ")); soma = soma + nota
nota = float(input("nota3 : ")); soma = soma + nota
nota = float(input("nota4 : ")); soma = soma + nota
nota = float(input("nota5 : ")); soma = soma + nota
...
#seria preciso repetir as linhas acima mais 45 vezes!

media = soma / 50
print('A media é: ', media)
```

Figura 1. Programa que calcula a média de 50 notas sem comando de repetição.

Veja que até é possível resolver o problema das médias de 50 notas sem usar repetição. No entanto o programa ficou bem grande e cheio de linhas repetidas ... tanto que omitimos 45 linhas de código para não encher essa página da apostila! Na próxima seção será apresentada a resolução deste problema com o uso do comando **while** e você perceberá que o programa ficará bem menor.

VI.2 A estrutura while

A estrutura **while** é utilizada no Python para permitir com que um bloco de comandos seja executado diversas vezes **enquanto uma condição for válida**. A Figura 2 ilustra o modelo de utilização do **while**.

```
while condição:
    comando1
    comando2
    …
    comandon
```

Figura 44. Modelo de utilização da estrutura *while*.

O **while** funciona da seguinte forma. Nós escrevemos while, depois especificamos uma condição (que irá resultar em True ou False) e depois dois pontos ":". O `while` vai realizar o **teste da condição no início** (isto é, antes da repetição ser disparada). Se o resultado do teste for verdadeiro (True), o comando ou o conjunto de comandos indentados abaixo do `while` **serão executados repetidas vezes** enquanto esta condição permanecer com o valor True. Estes comandos são chamados de "comandos subordinados ao while".

Para facilitar a compreensão observe na Figura 3 a maneira de implementar o programa que calcula a média de 50 notas agora com o uso da instrução **while**.

```
#Soma 50 números usando while – assim fica bonito!
print('Digite 50 notas: ')

#recebe 50 notas e acumula a soma dessas notas
soma = 0
i = 0
while i < 50:
    nota = float(input("nota : "))
    soma = soma + nota
    i = i + 1

media = soma / 50 #calcula e imprime a média
print('A média é: ', media)
```

Figura 3. Programa que calcula a média de 50 notas com o comando *while*.

No programa acima, os comandos destacados em vermelho negrito (**while** e seus comandos subordinados) serão executados 50 vezes, tornando possível somar as notas dos 50 alunos, para depois calcular a média da turma. Todo "segredo" da execução da repetição encontra-se no uso da variável "i" como **variável controladora** do comando **while**. Vamos simular a execução do programa, para que seu funcionamento possa ser melhor compreendido.

- Na linha 5 o valor 0 é atribuído à variável "soma" (ela será usada para armazenar a soma das notas dos 50 alunos). Na linha 6, "i" é inicializado com o valor 1. Nesse programa, a variável "i" é utilizada como variável controladora do comando **while**.

- As linhas 7 a 10 são as linhas que serão executadas 50 vezes. Mas como isso é possível? Da seguinte forma: quando o computador "vê" o comando **while** da linha 7 ele sabe que deve fazer um teste lógico. Nesse caso, o teste é o seguinte: "**o conteúdo de "i" é menor ou igual a 50?**" Note que, na primeira vez, a resposta será True, pois "i" foi inicializado com o valor 0. Com isto, o Python "entra no while", ou seja, decide executar os comandos indentados abaixo do while (linhas 8, 9 e 10), que são as instruções subordinadas ao **while**.

- Na linha 8, o valor de uma nota é solicitado via teclado e armazenado na variável "nota". Em seguida, na linha 9, este valor é somado ao conteúdo da variável "soma". A linha 10 possui uma instrução absolutamente necessária ao correto funcionamento deste programa: **o valor da variável "i" é incrementado em uma unidade**.

- Quando a execução do programa alcança a linha 10 (i = i + 1), que é o último comando subordinado ao while, **o controle é retornado para o while da linha 7**. Novamente será realizada a avaliação do comando while: "**o conteúdo de "i" é menor ou igual a 50?**" Na segunda vez, a resposta será True, pois "i" conterá o valor 2. Com isto, o Python "entra no while" novamente, ou seja, decide executar os comandos indentados abaixo do while (linhas 8, 9 e 10).

- Veja que esta repetição será realizada por 50 vezes, até que, na 51ª vez em que o teste do **while** for realizado, o valor de "i" será igual a 50. Neste momento, o teste `i < 50` resultará em False e o programa não executará as linhas 8, 9 e 10. Ao invés disso, o controle pulará para a linha `media = soma / 50` (cálculo da média). Depois executará a linha seguinte (a última do programa), com o `print` que imprime a média. Em seguida, o programa encerrará sua execução.

Algumas considerações importantes sobre o uso do comando while:

- No jargão da área de programação, uma estrutura de repetição definida em um programa é chamada de **laço** (ou *loop*, em Inglês). Cada repetição realizada no laço é chamada de **iteração**.

- É preciso utilizar uma ou mais variáveis controladoras para manter o laço enquanto isso for desejado.

- A variável ou variáveis controladoras, deverão ter o seu valor atribuído através de um comando de leitura ou atribuição antes da execução do comando `while`. Caso contrário o programa nunca entrará no laço (isto ocorre porque o comando `while` é um comando de repetição com teste no início).

- A variável controladora deverá ter o seu conteúdo modificado de alguma forma dentro do laço caso contrário o programa entrará em **loop infinito** (ou seja, os comandos subordinados ao `while` ficarão sempre em execução, pois o teste do `while` resultará sempre em True). O loop infinito é um dos erros mais "feios" da programação! Ele deixa um programa travado em um laço, fazendo a CPU trabalhar feito doida, consumindo recursos do computador.

- Os comandos subordinados ao `while` devem estar indentados, abaixo do `while`.

- As condições avaliadas pelo comando **while** também podem ser tão complexas como aquelas utilizadas pelo comando **if**. Exemplo:
```
while (i > 0) and ((j < 50) or (k == 10)) do
```

Repetição é preciso!

No primeiro exemplo do presente capítulo, vimos que o programa que calcula a média dos 50 alunos pode até ser resolvido sem o uso de um comando de repetição, mas isto nos obrigaria a escrever muitas linhas de código desnecessariamente. Porém, há outros tipos de problema que **exigem** o uso do comandos de repetição (ou seja, problemas que não podem ser resolvidos sem o emprego de algum comando de repetição).

Veja alguns exemplos:

- Ler o valor de N como entrada e calcular: H =1 + (1 / 2) + (1 / 3) + ... + (1 / N)
- Contabilizar o resultado de um concurso com mais de 200.000 participantes.
- Contabilizar a renda média do brasileiro a partir dos dados do Censo.

VI.3 Incremento e Decremento

Observe o exemplo a seguir e a sua solução utilizando o comando `while`.

EXEMPLO VI.2 Escreva um programa que imprima a palavra 'ENCE' e ' 15 vezes na tela usando o comando while.

RESOLUÇÃO:

```
x=0
while (x < 15):
    print('ENCE')
    x = x + 1
```

```
>>> %Run ence15x.py
ENCE
ENCE
ENCE
ENCE
ENCE
ENCE
ENCE
ENCE
ENCE
ENCE
ENCE
ENCE
ENCE
ENCE
ENCE
```

Neste programa, estamos usando a variável "i" para controlar o laço. Veja que a cada iteração do laço, seu valor é incrementado em 1 (instrução `i = i + 1`). Esse tipo de instrução – incrementar o valor de uma variável – é tão comum em programas que possui um nome, ela se chama **incremento**. Além de um nome específico, o incremento também possui uma sintaxe alternativa, mais compacta e preferida pelos programadores: `i = i + 1`, também pode ser escrito como **`i += 1`**. Sendo assim, poderíamos reescrever o programa que imprime ENCE 15 vezes na tela da seguinte maneira:

```
x=0
while (x < 15):
    print('ENCE')
    x += 1
```

Na verdade, podemos utilizar a sintaxe compacta para incrementar uma variável em qualquer valor, não apenas 1. Veja:

```
k += 5          #soma 5 ao valor corrente de "k"

v1 += 100       #soma 100 ao valor corrente de "v1"

soma += nota    #soma o valor armazenado em "nota"
                #ao valor corrente de "soma"
```

Além do incremento, existe também o **decremento**, que significa subtrair algum valor de uma variável. O decremento também tem uma sintaxe alternativa: `i = i - 1` também pode ser escrito como `i -= 1`. Veja mais alguns exemplos:

```
sal -= 50       #subtrai 50 do valor de "sal" em 50

temp -= x       #subtrai x do valor de "temp"
```

Por fim, é importante mencionar que também é possível utilizar a sintaxe curta para definir operações de multiplicação, divisão e potência. Veja os exemplos abaixo:

```
w *= 4        # faz "w" receber seu valor atual vezes 4

u /= 3        # faz "u" receber seu valor atual dividido por 3

z **= 2       # faz "z" receber o quadrado de seu valor atual

t %= 2        # faz "t" receber o resto da divisão de seu
              # valor atual por 2

r //= 2       # faz "r" receber o quociente da divisão de seu
              # valor atual por 2
```

VI.4 Problemas resolvidos

Esta seção apresenta uma série de exercícios resolvidos que demonstram a utilização do comando de repetição **while**. Digite os exemplos, execute com diferentes entradas e, se for preciso, faça simulações usando lápis e papel para entender claramente a situação das variáveis dentro de cada iteração do laço.

EXEMPLO VI.3: A conversão de graus Celsius para graus Fahrenheit é obtida pela seguinte fórmula:

F = C * 1,8 + 32

Elabore um algoritmo que calcule e imprima uma tabela de equivalência entre graus Celsius e Graus Farenheit, conforme ilustra a figura abaixo. A temperatura inicial em graus Celsius deve ser igual a –20ºC e a temperatura final igual a +100ºC. A escala da tabela em graus Celsius deve variar de 10 em 10.

-20 ºC	**-4 ºF**
-10 ºC	**14 ºF**
...	**...**
100 ºC	**212 ºF**

RESOLUÇÃO:

```
c = -20
print('* * * Tabela de conversão de graus Celsius para graus Fahrenheit')
while c <= 100:
    f = c  *1.8 + 32
    print(c,  '°C ----> ', f, '°F')
    c += 10

print('FIM!!!')

>>>
```

```
* * * Tabela de conversão de graus Celsius para graus Fahrenheit
-20 °C ---->  -4.0 °F
-10 °C ---->  14.0 °F
0 °C ---->  32.0 °F
10 °C ---->  50.0 °F
20 °C ---->  68.0 °F
30 °C ---->  86.0 °F
40 °C ---->  104.0 °F
50 °C ---->  122.0 °F
60 °C ---->  140.0 °F
70 °C ---->  158.0 °F
80 °C ---->  176.0 °F
90 °C ---->  194.0 °F
100 °C ---->  212.0 °F
FIM!!!
```

EXEMPLO VI.4: Elabore um programa onde o usuário possa digitar diversos números reais positivos via teclado. Quando o usuário digitar –1 a entrada deve encerrar e deve-se imprimir a soma dos números digitados e encerre o programa.

RESOLUÇÃO:

```
print("somando números")
print("===============")
print()

# laço que vai recendo os números e somando
# n é inicializado com 0 apenas para entrarmos ao menos 1x no laço

n = 0       #receberá cada número digitado pelo usuário
soma = 0    #acumulará a soma
while n != -1:
    print('Digite um número real positivo (ou digite -1 para encerrar) ')
    n = float(input())

    if (n > 0):    #atualiza a soma apenas se n for positivo
        soma += n

print('A soma dos números digitados é: ', soma)
```

Obs.: Neste exemplo **–1** é chamado de **flag de saída**. Esse é mais um jargão da área de programação. Significa o valor que faz o laço encerrar (ou seja, que que faz o programa sair do laço).

EXEMPLO VI.5: Elabore um programa que leia vários números inteiros e informe quantos números entre 100 e 500 foram digitados. Quando o valor 0 for lido, o programa deverá encerrar a sua execução (dessa vez nosso flag de saída é 0)

RESOLUÇÃO:

```
total = 0  # usada para contabilizar o total de números entre 0 e 100

# pede o 1° número fora do laço
# se o usuário digitar 0, o programa nem entrará no laço
print('Digite um número inteiro ou 0 para sair.')
a = int(input())

while (a != 0):
    if (a >=100) and (a <=500):
        total += 1

    print('Digite um número inteiro ou 0 para sair.')
    a = int(input())

print('total de números entre 100 e 500 = ', total)
```

EXEMPLO VI.6: **Entrar com o nome e idade de várias pessoas e imprimir quantas têm idade maior ou igual a 65 anos. O programa deve ser executado enquanto o usuário desejar.**

RESOLUÇÃO:

```
ficar = 'S'       # usada para controlar o laço enquanto o usuário quiser
total_acima_65 = 0 # contabiliza quantos têm 65 anos ou mais

while (ficar == 'S' or ficar == 's'):
    nome = input('Digite o nome: ')
    idade = int(input('Digite a idade: '))

    if idade >= 65:
        total_acima_65 += 1

    ficar = input('Deseja digitar mais dados (S=Sim, N=Nao)?: ')

print('total de idosos = ', total_acima_65)
```

EXEMPLO VI.7: **Dado um país A com 5.000.000 de habitantes e uma taxa de natalidade de 3% ao ano e um país B com 7.000.000 de habitantes e uma taxa de natalidade de 2% ao ano, calcular e imprimir o total de anos necessários para que a população do país A ultrapasse a população do país B.**

RESOLUÇÃO:

```
A = 5e6      # população do país A = 5.000.000
B = 7e6      # população do país B = 7.000.000
anos = 0     # número de anos p/ população de A ultrapassar a de B
```

```
while (A <= B):
    A = A * 1.03
    B = B * 1.02
    anos += 1

print('A será mais populoso que B dentro de ', anos, ' anos.')


>>>
A será mais populoso que B dentro de 35 anos.
```

EXEMPLO VI.8: Elabore um programa que leia o valor de N como entrada e calcule e imprima o valor de H, sendo:

H = 1 + (1 / 2) + (1 / 3) + ... + (1 / N).

Observações:

- *Para resolver um exercício de somatório é necessário usar sempre um ou mais comandos de **repetição**.*

- *Vocês devem observar os termos da expressão e identificar "**o que**" está variando de termo para termo e "**qual é o comportamento**" desta variação. As coisas que variam sempre devem ser implementadas na forma de variáveis no programa Python.*

    - *Exemplos de "coisas" que podem variar nas expressões: numerador, denominador, expoente, sinal (olhem para os termos e notem...).*

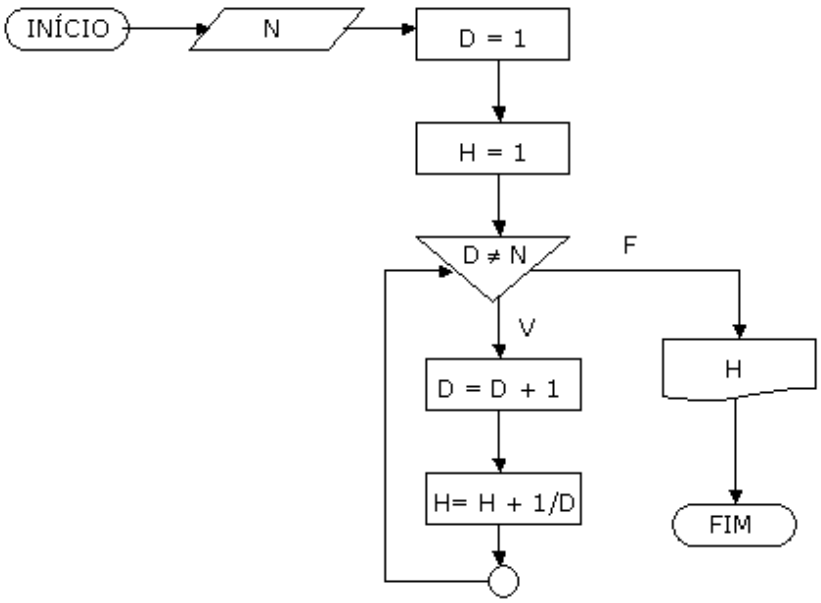
RESOLUÇÃO:

```
# PASSO 1: recebe o valor de N via teclado
print('Digite o valor de N: ')
N = int(input())

# PASSO 2: calcula H usando um laço
H = 1  # receberá o valor da série H
D = 1  # denominador -> é o cara que varia na série

while (D != N):
    D = D + 1
    H += 1 / D

print('H = ', round(H, 2)) # PASSO 3: imprime o valor calculado de H
```

Esse somatório é bastante simples:
- ***O numerador é sempre 1 em cada termo.***
- ***O denominador (D) começa em 1 e varia de 1 em 1 em cada termo.***

- ***A série tem N termos. Se N=1, o valor de H é 1. Se N ≠ 1, precisamos montar um LOOP para determinar o valor de H.***

Veja que a fórmula para o cálculo do valor de H é inteiramente dependente do valor de N. Caso o usuário escolha o valor 1, o somatório que determina H só teria um termo (H = 1). Se o usuário digitar 2, o cálculo de H é dado por H = 1 + ½. Se o usuário digitar 5 o cálculo de H passa a ter 5 termos: H = 1 + ½ + 1/3 + ¼ + 1/5. A título de curiosidade, a seguir ilustra-se o algoritmo no formato de fluxograma para a resolução do problema.



EXEMPLO VI.9: Construa um programa que calcule e escreva o valor de S:

S = (1/1) + (3/2) + (5/3) + (7/4) + ... + (99/50).

Esse somatório também é simples:

- ***o numerador começa em 1 e termina em 99, variando de 2 em 2***
- ***o denominador começa em 1 e termina em 50 variando de 1 em 1***
- ***veja que a série tem 50 termos (valor fixo – não há entrada no programa).***

RESOLUÇÃO:

```
# inicializa as variáveis
S=0
NUM=1
DEN=1
```

```
# laço simples que implementa a soma
# veja que estou controlando o denominador com a variável DEN
# e o numerador com a variável NUM
while (DEN <= 50):
    S += (NUM / DEN)
    DEN += 1
    NUM += 2

#imprime o resultado
print('S = ',S)

>>>
S =  95.5007946616706
```

EXEMPLO VI.10: Construa um programa que receba o valor de N como entrada e calcule o valor de N!

RESOLUÇÃO:

```
#lê n
print('CALCULO DO FATORIAL')
print('*******************')
n = int(input('Digite n (inteiro positivo): '))

#calcula n!
fat = 1
i = 1
while (i <= n):
    fat = fat * i
    i = i + 1

#imprime o resultado
print('n!  = ', fat)
```

Observe que quando "n" é zero, o programa não entra no laço e determina o valor de "fat" como 1, o que é correto, pois 0! = 1. Em qualquer outro caso o programa entra no laço.

EXEMPLO VI.11: Foi feita uma pesquisa de audiência de canal de TV aberta em várias casas de uma certa cidade, em um determinado dia. Para cada casa visitada, foi anotado o número do canal onde a TV estava ligada (existem apenas sete canais: 2, 4, 6, 7, 9, 11 e 13) e a quantidade de pessoas que estavam assistindo a esse canal (para simplificar, considere que existia apenas um televisor em cada residência). Elabore um programa que:

- **Leia os dados desta pesquisa a partir do teclado (ou seja leia pares de valores CANAL e QUANTIDADE DE PESSOAS). Quando o usuário digitar o valor "0" para um canal, a leitura deve ser encerrada.**

- **Ao final da leitura, o programa deve imprimir o total de pessoas que assistiu a cada um dos 6 canais no dia da pesquisa. Deve imprimir também a porcentagem de audiência de cada canal (ver Figura abaixo "N" e "X" devem ser trocados, respectivamente, pelo número de pessoas e a porcentagem de audiência).**

CANAL 2 N_1 PESSOAS - X_1% DE AUDIÊNCIA
…
…
CANAL 13 N_6 PESSOAS - X_6% DE AUDIÊNCIA

***OBS:** É importante estudar e entender a resolução do problema, pois ele ilustra um exemplo simples de programa para tabular de dados de uma pesquisa.*

DICA PARA RESOLUÇÃO DE UM PROBLEMA DESTE TIPO: vocês podem, por exemplo, anotar no papel os passos, as etapas que vocês executariam caso fossem resolver o programa manualmente (ou seja, somando a audiência de cada emissora num caderno ou usando uma calculadora). Depois disso, vocês traduzem o método de resolução "manual" para uma sequência de comandos Python.

RESOLUÇÃO:

```
# ----------------------------------------
#PASSO 1: INICIALIZA AS VARIÁVEIS
# ----------------------------------------

# estas variáveis recebem o total de pessoas que assistiu cada canal
# a sintaxe abaixo é permitida pelo Python para atribuir o mesmo valor a todas
# (nesse caso, valor 0)
c2 = c4 = c6 = c7 = c9 = c11 = c13 = 0

N = 0 #total geral de pessoas contabilizadas pela pesquisa


# ----------------------------------------
#PASSO 2: LOOP QUE LÊ E CONTABILIZA OS DADOS DA PESQUISA
# ----------------------------------------
print('DIGITE OS DADOS DA PESQUISA')
print('**************************')

num_canal = 999 #inicializa com um valor qualquer ≠ 0 só para entrar no while

while (num_canal != 0):
    print()
    num_canal = int(input('NUMERO DO CANAL: '))
    if num_canal != 0:
        qtd_pessoas_na_casa = int(input('QUANT. PESSOAS NA CASA: '))

    #atualiza a audiência do canal usando if-elif
    if num_canal == 2:
```

```
        c2 += qtd_pessoas_na_casa
        N += qtd_pessoas_na_casa
    elif num_canal == 4:
        c4 += qtd_pessoas_na_casa
        N += qtd_pessoas_na_casa
    elif num_canal == 6:
        c6 += qtd_pessoas_na_casa
        N += qtd_pessoas_na_casa
    elif num_canal == 7:
        c7 += qtd_pessoas_na_casa
        N += qtd_pessoas_na_casa
    elif num_canal == 9:
        c9 += qtd_pessoas_na_casa
        N += qtd_pessoas_na_casa
    elif num_canal == 11:
        c11 += qtd_pessoas_na_casa
        N += qtd_pessoas_na_casa
    elif num_canal == 13:
        c13 += qtd_pessoas_na_casa
        N += qtd_pessoas_na_casa

# ----------------------------------------
# PASSO 3: IMPRIME OS RESULTADOS
# ----------------------------------------
print()
print('RESULTADOS DA PESQUISA:')
print('**********************')
print()

# IMPRIME CANAL POR CANAL (pois ainda não sabemos usar listas...}
# usamos end="" no print do canal,
# para que o próximo print saia do lado e não embaixo

#CANAL 2
pct_audiencia = (c2 / N) * 100
print('- CANAL 2 : ', c2, ' PESSOA(S) - ', end="")
print(round(pct_audiencia,2), '% DE AUDIENCIA')

#CANAL 4
pct_audiencia = (c4 / N) * 100
print('- CANAL 4 : ', c4, ' PESSOA(S) - ', end="")
print(round(pct_audiencia,2), '% DE AUDIENCIA')

#CANAL 6
pct_audiencia = (c6 / N) * 100
print('- CANAL 6 : ', c6, ' PESSOA(S) - ', end="")
print(round(pct_audiencia,2), '% DE AUDIENCIA')

#CANAL 7
pct_audiencia = (c7 / N) * 100
print('- CANAL 7 : ', c7, ' PESSOA(S) - ', end="")
print(round(pct_audiencia,2), '% DE AUDIENCIA')

#CANAL 9
pct_audiencia = (c9 / N) * 100
print('- CANAL 9 : ', c9, ' PESSOA(S) - ', end="")
print(round(pct_audiencia,2), '% DE AUDIENCIA')

#CANAL 11
pct_audiencia = (c11 / N) * 100
print('- CANAL 11 : ', c11, ' PESSOA(S) - ', end="")
```

```
print(round(pct_audiencia,2), '% DE AUDIENCIA')

#CANAL 13
pct_audiencia = (c13 / N) * 100
print('- CANAL 13 : ', c13, ' PESSOA(S) - ', end="")
print(round(pct_audiencia,2), '% DE AUDIENCIA')
```

***OBS:** Quando você aprender a trabalhar com listas (próximo capítulo), verá que é possível resolver esse problema de uma forma muito mais elegante, que utiliza bem menos linhas de código.*

VI.5 A estrutura *for-range()*

Assim como o `while`, o comando `for` também serve para implementar a repetição de blocos de código. Ele existe em praticamente todas as linguagens de programação e, por ser mais prático do que o `while`, é normalmente escolhido pelos programadores em situações onde deseja-se executar um conjunto de comandos por um **número fixo de vezes**.

VI.5.1 – Uso básico do for-range()

Na linguagem Python o `for` possui uma característica bastante singular (e relevante!): ele pode iterar apenas sobre coleções. Neste capítulo, mostraremos como utilizar o `for` para iterar sobre coleções criadas com a função `range()` e a partir do capítulo seguinte, veremos como trabalhar com outras coleções. A Figura 4 ilustra **o modelo mais simples** de utilização do **for-range()** (existem outros, mas começaremos com esse):

```
for v in range(n):
    comando1
    comando2
    …
    comandon
```

Figura 4. Modelo de utilização da estrutura *for-range()*

Nesta sintaxe:

- "v" é o nome da variável de controle. Essa variável será tratada como uma variável inteira.
- "n" corresponde ao tamanho da sequência que desejamos gerar. Quando utilizamos **range(n)**, o resultado será uma sequência de números inteiros que começa com *0* e vai até *n-1*: {*0*, *1*, *2*, ..., *n-1*}
- Durante a execução do laço, a variável de controle assume o valor inicial da sequência na primeira iteração (neste caso, o valor 0). A cada iteração, o valor da variável de

controle será automaticamente modificado para o próximo valor da sequência. O laço termina, quando o seu valor atingir o valor final, ou seja, *n-1*.
    - Com isso, o `for` utilizado em conjunto com `range(n)` realizará exatamente *n* iterações.
- A linha com o comando **for-range()** deve terminar com dois pontos ":".

- Os comandos subordinados ao `for` devem estar indentados.

O exemplo da Figura 5 ilustra a forma de implementar um programa que calcula a média de 50 notas com o uso do comando **for-range()**. Compare esta implementação com aquela que utiliza o comando **while**.

```
#Soma 50 números usando for – sem dúvida, é bonito!
print('Digite 50 notas: ')

#recebe 50 notas e acumula a soma dessas notas
soma = 0
for i in range(50):
    nota = float(input("nota : "))
    soma = soma + nota

media = soma / 50 #calcula e imprime a média
print('A média é: ', media)
```

Figura 5. Programa que calcula a média de 50 notas com o comando de repetição *for*.

O programa da Figura 5 funciona da seguinte forma:

- Na linha 5 o valor 0 é atribuído à variável "soma" (ela será usada para armazenar a soma das notas dos 50 alunos);

- As linhas 6 a 8 formam o laço que será executado 50 vezes.

- Quando a linha 6 é processada, determina-se que o Python realizará uma iteração sobre a sequência {0, 1, 2, ..., 49} – gerada pela função **range(50)**.

- Na primeira iteração, "i" (variável de controle) é igual a 0 e o bloco de código subordinado ao *for* (linhas 7 e 8). Quando alcançar a linha 8, o controle será retornado para a linha 6. Assim começará a segunda iteração, em que "i" passará automaticamente para o valor 1, que é o próximo da sequência. Mais uma vez os comandos subordinados ao `for` serão executados (linhas 7 e 8). Isso será feito repetidamente até o valor de "i" chegar no último da sequência (nesse caso, 49). Então uma última iteração acontecerá e a repetição será encerrada.

- Quando o *loop* é encerrado, o controle passará para a linha 10 (cálculo da média). Depois executará a linha 11 (o valor da média será impresso) e em seguida encerrará sua execução.

A Figura 6 apresenta um programa que usa um laço montado com **for-range()** para imprimir 15 vezes a palavra LINHA, seguida do valor da variável *"k"* (variável de controle do **for**). A Figura 7 exibe a saída do programa. Observe que o valor de "k" muda (aumenta em 1) para cada iteração do *loop*, uma vez que a função `range()` gera a sequência {0, 1, 2, ..., 14}.

```
for k in range(15):
      print('LINHA ',k)
```

Figura 6. Programa que imprime 15 linhas com o uso do comando *for*.

```
>>> %Run imp15linhas_for.py
LINHA  0
LINHA  1
LINHA  2
LINHA  3
LINHA  4
LINHA  5
LINHA  6
LINHA  7
LINHA  8
LINHA  9
LINHA  10
LINHA  11
LINHA  12
LINHA  13
LINHA  14
```

Figura 7. Resultado da execução do programa que imprime 15 linhas

VI.5.2 – definindo os limites da sequência gerada pelo range()

A função `range()` serve para gerar uma sequência de números inteiros que é geralmente utilizada para ser percorrida por um comando `for`. Na seção anterior, vimos que utilizando `range(n)` serve para gerar a sequência {*0, 1, 2, ..., n-1*}. Esta sequência pode ser utilizada para definir um laço com *n* iterações, em que a variável de controle assumirá o valor 0 na primeira iteração e o valor *n-1* na última.

Entretanto, a função `range()` é sofisticada, possibilitando a geração de outros tipos de sequência de números inteiros. Por exemplo, sequências que começam com outro valor diferente de 0 e até mesmo sequências decrescentes.

Vamos aprender a fazer isso? O segredo é entender a sintaxe completa da função `range()`, apresentada na Figura 8. Ela aceita que até 3 parâmetros sejam utilizados para definir uma sequência: início, fim e incremento.

```
range(início, fim, incremento)
```

Figura 8. Sintaxe completa da função range()

- **início**: número inicial da sequência (opcional). Caso seja omitido, o valor 0 é assumido.

- **fim**: a sequência será gerada até, **mas sem incluir**, o número especificado neste parâmetro (único parâmetro obrigatório). **ENTÃO MUITO CUIDADO!!!** A sequência nunca irá incluir o valor que você especificar no parâmetro fim. Ela sempre vai parar 1 valor antes.

- **incremento**: diferença entre cada número na sequência (opcional). Se omitido, o valor 1 será utilizado.

Quando fazemos `range(50)`, na verdade estamos utilizando a função passando apenas o parâmetro fim, com valor de 50. Então assume-se que o início da sequência é 0 e o incremento é 1 (já que o início e o incremento são opcionais e estes são seus valores padrão). Por isso, a sequencia {0, 1, 2, ..., 49} é gerada.

A Figura 9 demonstra diversas formas de utilização do range(), através da definição de diferentes valores para os parâmetros "início", "fim" e "incremento". O resultado da execução é apresentado na Figura 10.

```
#diferentes sequências geradas por range() e exploradas pelo for
print()
print('* * imprimindo de 0 a 3')
for i in range(4):
    print(i)

print()
print(' * * imprimindo de 100 a 105')
for i in range(100, 106):
    print(i)

print()
print(' * * 0 a 15, usando 5 como incremento')
for i in range(0, 16, 5):
    print(i)

print()
print(' * * ordem reversa: 5, 4, 3, 2, 1')
for i in range(5, 0, -1):
    print(i)
```

Figura 9. Diferentes sequências usadas com o range()

```
* * imprimindo de 0 a 3
0
1
2
3

* * imprimindo de 100 a 105
100
101
102
103
104
105

* * 0 a 15, usando 5 como incremento
0
5
10
15

* * ordem reversa: 5, 4, 3, 2, 1
5
4
3
2
1
```

Figura 10. Resultado da execução do programa da Figura 9

A seguir, algumas considerações importantes sobre o uso do comando *for*:

- O comando **for** executa um laço por um **número fixo** de vezes, que será igual ao número de elementos da sequência gerada pelo **range()**.

- O programador não tem a necessidade de mexer no valor da variável de controle dentro do laço (na verdade, ele **não deve** fazer isso). O valor da variável de controle é modificado **automaticamente** para o próxima da sequência no início de cada iteração do *loop*. Na primeira iteração do *loop* o valor da variável de controle será igual ao primeiro valor da sequência. Na segunda iteração será igual ao segundo valor da sequência. E na última, o valor será igual ao valor final da sequência.

- O início, fim e incremento do **range()** devem ser **valores inteiros** podendo ser valores positivos, negativos ou zero. Não é possível utilizar um valor real para esses parâmetros.

- Não é preciso atribuir nenhum valor para a variável controladora antes da execução do comando *for* (ao contrário do que ocorre com o comando *while*)

- Os comandos subordinados ao *for* devem estar localizados abaixo dele, devidamente indentados com 4 espaços.

- Com o uso do *for* o programa **nunca entra** em *loop* infinito, pois ele será repetido por um número fixo de vezes. O programa só corre risco de entrar em *loop* infinito se o programador alterar de forma equivocada o valor da variável de controle dentro dos comandos subordinados ao *for*.

    - Embora, o alterar o valor da variável de controle seja algo permitido pelo Python e outras linguagens e a despeito do fato de que programadores mais experientes eventualmente façam isso em seus programas (normalmente com o intuito de aumentar a eficiência, ou seja, tornar o programa mais rápido), neste curso introdutório recomendamos fortemente que você evite fazer isso!

VI.6 Quando usar o for? Quando o while?

Tudo que é feito com a estrutura **for-range()** pode ser implementado com a instrução **while**. Mas a recíproca não é verdadeira: existem certos tipos de laço que só podem ser implementados com **while**. Resumidamente, o que ocorre é o seguinte:

- Se você vai implementar um laço que será executado por um número **previsível** (fixo) de vezes, você **pode utilizar** a estrutura **for-range()** (já que ela é mais simples e não nos obriga a incrementar a variável de controle!). Alguns exemplos de programas que podem ser implementados com o comando *for*:

    - Programa que recebe as notas de 50 alunos e calcula a média (o laço é executado por um número fixo de vezes: 50).

    - Programa que determina a soma se uma série contendo N termos (basta executar o N vezes).

- Se você vai implementar um laço que será executado por um número **imprevisível** de vezes, **não é possível** utilizar a estrutura **for-range()**. Alguns exemplos:

    - Programa que deve ser mantido em execução enquanto o usuário desejar (por exemplo, até que usuário digite –1. Não podemos prever em que iteração o usuário fará isso!).

    - Programa que calcula o M.D.C. de dois inteiros pelo método de Euclides (não sabemos em que iteração que o resto da divisão será igual a zero – consulte: http://clubes.obmep.org.br/blog/sala-de-estudos-algoritmo-de-euclides-para-determinacao-de-mdc/diagramas/

VI.7 O for reverso

No último exemplo da Figura 9, mostramos que é possível definir uma sequência decrescente no comando `range()`, bastando para isso usar um "incremento" negativo e definir o o parâmetro "fim" com um valor maior do que o do "início".

Uma forma ainda mais simples de fazer o `for` "reverso" é utilizando a função **reversed()**. Veja como é simples no exemplo a seguir: você especifica uma sequência normal e a função `reversed()` a inverte pra você!

```
print('* * ordem reversa: 5, 4, 3, 2, 1')
for i in reversed(range(1, 6)):
    print(i)

>>>
* * ordem reversa: 5, 4, 3, 2, 1
5
4
3
2
1
```

VI.8 Comandos break e continue

Agora serão introduzidos dois comandos muito úteis na implementação de laços: **break** e **continue**. Ambos podem ser utilizados dentro do **while** ou dentro do **for**. O comando **break** pode ser utilizado quebrar um laço "na marra", passando o fluxo de execução do programa para a linha que estiver localizada imediatamente depois do fim do bloco de comandos subordinados ao laço. Por sua vez, o comando **continue** serve para quebrar uma iteração, mas não o laço propriamente dito. Mais claramente: sempre que o continue é executado, o fluxo de execução do programa é automaticamente desviado para a linha que contém o comando **while** ou **for**. A seguir apresenta-se um exemplo do funcionamento destes comandos.

```
print('-------------------------------------')
print('(1)-Exemplo de while com break:')
n = -1
while (n < 21):
    n = n+1
    if n % 2 != 0:
        break #quebra o laço se n for ímpar...

    print(n)

print('fim do while com break...'); print()


print('-------------------------------------')
print('(2)-Exemplo de while com continue:')
n = -1
while (n < 21):
    n = n+1
    if n % 2 != 0:
```

```
        continue # quebra a iteração se n for ímpar
                 # mas o laço não é quebrado

    print(n)

print('fim do while com continue...')


>>>

-------------------------------------
(1)-Exemplo de while com break:
0
fim do while com break...

-------------------------------------
(2)-Exemplo de while com continue:
0
2
4
6
8
10
12
14
16
18
20
fim do while com continue...
```

VI.9 Laços aninhados

Muitos problemas só podem ser resolvidos com a utilização de *loops* aninhados, o que significa um laço dentro de outro laço. Melhor explicando: utilizar um *for* dentro do outro, um *while* dentro de um *for*, um *while* dentro de um *while* etc. Em muitos casos, é preciso até mesmo utilizar mais de 2 laços aninhados (ex.: *while* dentro de um *for* dentro de outro *for*).

A Figura 11 apresenta um exemplo de programa com *loops* aninhados. Este programa solicita com que o usuário digite um número entre 1 e 10 e, em seguida, mostra a "tabuada" deste número.

O programa é mantido em execução até que o usuário deseje encerrar (uma pergunta é feita sempre que uma tabuada é mostrada). Os dois laços do programa funcionam da seguinte forma: o laço externo (*while*) é usado para manter o programa em execução enquanto o usuário desejar. O laço interno (*for*) computa e mostra na tela a tabuada do número entrado pelo usuário.

```
resp = 'S'

# laço externo com while
# usado para manter o programa em execução
while resp == 'S' or resp == 's':

    n = int(input('Digite um numero entre 1 em 10: '))

    if n >= 1 and n <= 10: # computa a tabuada apenas se
                           # usuário digitou valor válido,
                           # ou seja, entre 1 e 10

        #o laço for computa e imprime a tabuada de n
        for i in range(1, 11):
            print(n, 'x', i, ' = ', n *  i)

        print()
        resp = input('Deseja mais um número (S=Sim, N=Nao)?')

print('OBRIGADO POR USAR O PROGRAMA TABUADA')
```

Figura 11. Programa Tabuada – Exemplo de *Loops* Aninhados.

VI.10 Problemas resolvidos

Esta seção apresenta uma série de exercícios resolvidos que exemplificam o uso do comando **for**. Alguns conceitos novos são apresentados em alguns dos exemplos.

EXEMPLO VI.11 Escreva um programa que imprima a palavra 'ENCE' 30 vezes na tela, usando o comando for.

RESOLUÇÃO:

```
for m in range(30): print('ENCE')
```

Este programa ilustra um exemplo de uma estrutra `for-range()` com **apenas um comando subordinado:** `print('ENCE')`. Neste caso é possível colocar o comando subordinado após os dois pontos. Isso também é possível com os comandos `while`, `if`, `elif` e `else`. Se eles possuem apenas um comando subordinado, você pode colocar esse único comando ao lado em vez de colocá-lo indentado abaixo (como mostrado abaixo):

```
for m in range(30):
    print('ENCE')
```

Nesse exemplo, a variável de controle se chama "m". A variável de controle pode ter qualquer nome, contanto que seja um nome válido (veja no capítulo 4 as regras para dar nomes a variáveis).

EXEMPLO VI.11 Escreva um programa que receba a nota de 20 alunos como entrada e que imprima qual delas é a maior.

RESOLUÇÃO:

```
MAX = 20   #número de valores que serão digitados

print('Programa Maior Nota')
print('====================')

print('Digite ', MAX, ' Notas:')

maior_nota = 0
for i in range(MAX):
    print('Nota', i+1, ':')
    nota = float(input())
    if (nota > maior_nota): maior_nota = nota

print('A maior nota digitada foi', maior_nota)
```

Observe que neste programa utilizamos a variável MAX (de valor 20) bem no início do programa para definir o limite superior do `range()` e, por consequência, o número de iterações do `for`. Desta maneira fica muito simples mudar o programa para que ele passe a aceitar como entrada 10 notas ou 50 notas (basta mexer no valor de MAX, especificado na linha 1).

EXEMPLO VI.12 Utilizando o comando for, construir um programa Python que leia um número inteiro positivo n como entrada e calcule e exiba na tela o valor de n! (ou seja, o fatorial de n).

Obs: o programa não deve aceitar um valor negativo como entrada.

```
#lê n
print('CALCULO DO FATORIAL')
print('*******************')
n = int(input('Digite n (inteiro positivo): '))

#calcula n!
fat = 1
for i in range(1, n+1):
    fat = fat * i

#imprime o resultado
print('n!  = ', fat)
```

Neste exemplo, um laço montado com "i" variando entre 1 e "n" (número digitado pelo usuário) é utilizado para o cálculo do fatorial. Para isso, bastou fazer `range(1, n+1)`. Se o usuário digitar o valor 0, pois geraria `range(1,1)` (sequência de 1 a 0 com incremento positivo, o que é impossível de ser gerado).

Um problema é que o usuário pode digitar números negativos. Se quisermos defender o nosso programa quanto a esse tipo de entrada, basta utilizar o `if`, como já fizemos algumas vezes. Veja abaixo:

```
#lê n
print('CALCULO DO FATORIAL')
print('********************')
n = int(input('Digite n (inteiro positivo): '))

if n >= 0:      # calcula n! apenas se valor não é negativo

    fat = 1
    for i in range(1, n+1):
        fat = fat * i

    #imprime o resultado
    print('n!  = ', fat)

else : print('o valor entrado não pode ser negativo')
```

EXEMPLO VI.13 Construir um programa PYTHON com dois comandos for aninhados para imprimir na tela a tabuada de 1,2,3,4,5,6,7,8,9 e 10.

```
for i in range(1, 11):
    print('Tabuada de ', i)
    for j in range(1, 11):
        print(i, 'x', j, '=', i * j)

    print()
```

EXEMPLO VI.14 Construir um programa Python que leia um número inteiro positivo *n* como entrada e determine se o número é primo ou não

Observações :

- **O programa não deve aceitar um valor negativo como entrada.**
- **O programa deve ser mantido em execução enquanto o usuário desejar.**
- **Para descobrir se um número é primo, basta dividi-lo sucessivamente do valor 2 até o valor de sua raiz quadrada. Se houver resto em todas as divisões, ele é primo.**

```
import math

print('PROGRAMA PRIMO')
print('==============')

encerrar = 'N'

while encerrar != 'X' and encerrar != 'x':
    n = int(input('Digite um número inteiro positivo: '))

    if (n <= 0):
        print('O número digitado não é positivo.')

    else:
        #aqui eu determino se "n" é primo!
        if n == 1: primo = False  #1 não é primo

        elif n == 2: primo = True #2 é primo

        else: #trata do 3 em diante
            primo=True #parte da hipótese de que é um primo
            for i in range (2, math.trunc(math.sqrt(n))+1):
                if (n % i) == 0:
                    primo = False
                #aqui eu já determinei! Rápido, não é?

        #imprime o resultado}
        if primo:
            print('---> ', n, ' é primo')
        else:
            print('---> ', n, ' não é primo')

      #pergunta se usuario quer encerrar}
    print()
    print('TECLE "X" PARA ENCERRAR OU OUTRA TECLA PARA CONTINUAR: ')
    encerrar = input()
```

Neste exemplo, um laço externo montado com o comando **while** mantém a rotina que recebe "n" e determina se ele é primo em execução até que o usuário digite a tecla "x" (tecla escolhida pelo programador para servir de flag de saída).

Para determinar se "n" é primo ou não, primeiro tratamos os casos especiais 1 (que não é primo) e 2 (que é um primo). Para os números 3 em diante, implementamos um **for-range()** que divide *n* por dois até o **primeiro inteiro acima de sua raiz**, valor obtido com `math.trunc(math.sqrt(n))+1`. Recorde que `math.trunc` (efetua o truncamento de um número) e `math.sqrt` (calcula a raiz quadrada de um númer) são funções do módulo 'math', apresentado no capítulo 4. Esse módulo é importado logo na primeira linha do programa.

Mas por que foi preciso usar a função `math.trunc`? Ela foi necessária porque o limite superior do **for** precisa ser um inteiro! Se você usar apenas `math.sqrt(n)`, verá que o Python acusará um **erro de tipo** (**TypeError**).

Exercícios propostos

Resolva os exercícios a seguir utilizando **while** ou **for**. Em alguns exercícios, não será possível utilizar apenas um comando de repetição e você terá que fazer uso de laços aninhados (`for` dentro de `for`, `while` dentro de `for`, `while` dentro de `while` etc.).

(1) Crie um programa que apresente a soma de todos os números ímpares entre 0 e 10.000.

(2) Crie um programa que leia o valor de um inteiro positivo *n* via teclado e depois imprima o quadrado de todos os números entre 1 e *n*, da forma mostrada a seguir:

```
Digite o valor de n: 13
1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
4 * 4 = 16
5 * 5 = 25
6 * 6 = 36
7 * 7 = 49
8 * 8 = 64
9 * 9 = 81
10 * 10 = 100
11 * 11 = 121
12 * 12 = 144
13 * 13 = 169
```

(3) A conversão da medida de distância em centímetros (cm) para polegadas (pol) é obtida pela seguinte fórmula:

POL = CM * 0,3937.

Elabore um programa que **calcule** e **imprima** uma tabela de equivalência entre centímetros e polegadas, cujo valor inicial em centímetros deve ser igual a 0cm e o valor final igual a 1000cm. A escala da tabela deve variar de 5 em 5cm.

(4) Escreva um programa onde o usuário entre com um número inteiro e o programa verifique se o número é par ou ímpar. O programa deve ser executado enquanto o usuário desejar.

(5) Altere o programa que calcula o fatorial de um número *n* com `while` (Exemplo VI.10), para que ele não aceite como entrada nenhum número menor do que zero ou maior do que 20.

(6) Crie um programa que utilize o comando `for` para imprimir os números de 1 a 15 de maneira **exatamente igual** à apresentada na imagem abaixo (incluindo as vírgulas, espaços em branco e colchetes).

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

DICA: utilize o parâmetro "end" da função print()

(7) Altere o programa anterior para que agora sejam impressos os números de 101 a 300.

(8) Crie um programa que leia o valor de "n" via teclado e depois imprima a palavra 'ENCE' "n" vezes da maneira ilustrada na figura abaixo.

```
Digite o valor de n: 12
ENCE
 ENCE
  ENCE
   ENCE
    ENCE
     ENCE
      ENCE
       ENCE
        ENCE
         ENCE
          ENCE
           ENCE
            ENCE
             ENCE
```

(9) Escreva um programa que permita com que o usuário digite diversos números inteiros. Quando ele digitar 0 (flag de saída) o programa deve exibir Sp e Si sendo que:
Sp = soma dos números pares digitados pelo usuário.
Si = soma dos números ímpares digitados pelo usuário.

(10) Escreva um programa que calcule e imprima o valor da seguinte soma:

S = (37 x 38) / 1 + (36 x 37) / 2 + (35 x 36) / 3 + ... + (1 x 2) / 37

(11) Escreva um programa que calcule e imprima o valor da seguinte soma:

S = (1000 / 1) – (997 / 2) + (994 / 3) – (991 / 4) + ...

(12) Construir um programa que leia o valor de N ($0 < N \leq 10$) a partir do teclado e calcule e escreva o valor do seguinte somatório:

S = (N! / 1) – (N! / 2) - (N! / 3) – (N! / 4) - ... - (N! / 25).

Obs: o programa não deve permitir a entrada de um valor de N fora de faixa $0 < N \leq 10$

(13) O número 3025 possui a seguinte característica:

- 30 + 25 = 55
- $55^2 = 3025$

Crie um programa que pesquise e imprima todos os números de quatro algarismos que apresentem tais características.

(14) Elabore um programa que leia 20 números via teclado e imprima:

- A soma dos números digitados.
- O maior número digitado.
- O menor número digitado.

(15) Elabore um programa que leia a matrícula e a nota final de 10 alunos de uma turma da disciplina Estatística Computacional. Em seguida imprima:

- A média das notas da turma.
- A matrícula e a nota do aluno com melhor desempenho (suponha que não tenha havido empate).

(16) Escrever um programa que leia os votos da eleição para síndico de um prédio. O programa deve ler os votos do teclado até que o valor "–99" seja digitado.

- Se o número **1** for digitado, deve-se computar 1 voto para o candidato **Sr. Rakesh**.
- Se o número **2** for digitado, deve-se computar 1 voto para o candidato **Sr. Zaki**.
- Se **outro número** for digitado, deve-se computar 1 **voto nulo** (suponha que não existam votos em branco).

Ao final do processamento, o programa deve exibir:

- A quantidade de votos em Rakesh.
- A quantidade de votos em Zaki.
- A quantidade de votos nulos.
- O nome do vencedor da eleição.
    - Se houver empate, considere que Rakesh foi eleito, por ele ser o candidato mais velho.
    - Se o número de votos nulos for maior do que o número de votos em Raksh e também maior do que o número de votos em Zaki, o sistema deve imprimir a seguinte mensagem "O número de votos nulos superou os votos dos dois candidatos. Não houve vencedor".

(17) Criar um programa que leia a idade e o sexo (M = Masculino e F = Feminino) de várias pessoas. Calcule e imprima a idade média, o total de pessoas do sexo feminino e o total de pessoas do sexo masculino. A entrada de dados deve encerrar quando se digita –1 para a idade (flag de saída).

Ignore um dado de entrada se o sexo digitado for diferente de 'M' ou 'F' ou se a idade digitada for menor que 0.

(18) Uma pesquisa que investigou renda e escolaridade possuía o seguinte questionário:

(a) Qual a sua escolaridade? ___ 1-Fundamental 2-Médio 3-Superior 4-Pós-Graduação (b) Qual a sua renda? _____

Crie um algoritmo que leia os dados de entrada da pesquisa via teclado (os dados devem ser lidos até o usuário digitar –1 para o código da escolaridade). Em seguida imprima:

- A média salarial das pessoas com nível superior
- O número de pessoas com Pós-Graduação
- O número de pessoas com nível médio que ganham mais de R$ 2.000 por mês.

(19) Construir um programa que descubra e imprima todos os números perfeitos menores ou iguais a 200.

Obs: um número natural n > 1 é perfeito se a soma de seus divisores, exceto ele próprio, é igual ao número. Ex: 1 + 2 + 3 = 6.

(20) Uma das maneiras de se conseguir a raiz quadrada de um número é subtrair do número os ímpares consecutivos a partir de 1, até que o resultado da subtração seja menor ou igual a zero. O número de vezes que se conseguir fazer a subtração é a raiz quadrada exata (resultado 0) ou aproximada do número (resultado negativo).

Exemplo: raiz de 16

16 – 1 = 15 – 3 = 12 – 5 = 7 – 7 = 0. A raiz de 16 é 4.

Faça um programa que obtenha a raiz de um número usando este método.

(21) Construir um programa que leia o valor de N a partir do teclado e calcule e escreva o valor do seguinte somatório:

$S = (N^{25} / 1) – (N^{24} / 2) + (N^{23} / 3) – (N^{22} / 4) + ... + (N / 25)$.

DICA: será preciso implementar um laço dentro de laço.

(22) Um arranjo de um conjunto de *n* objetos, em dada ordem, é chamado de **permutação** dos objetos (tomados todos ao mesmo tempo). Um arranjo de quaisquer $r \leq n$ destes objetos, em dada ordem é chamado de *r*-permutação ou permutação dos *n* objetos tomados *r* a *r*.

Exemplo: considerando as letras A, B, C e D, tem-se que:

- BAD, ADB, CBD e BCA são exemplos de permutação das 4 letras tomadas 3 a 3
- BDCA, DCBA, ACDB são exemplos de permutações das 4 letras tomadas 4 a 4
- AD, CB, DA e BD são exemplos permutações das 4 letras tomadas 2 a 2.

O número (quantidade) de permutações de *n* objetos tomados *r* a *r* pode ser calculado através da seguinte fórmula:

$$\mathbf{P(n,r) = n! / (n – r)!}$$

Elabore um programa que receba como **entrada** os valores de ***n*** e ***r*** e calcule e imprima a quantidade de permutações de n objetos tomados r a r (*basta usar a fórmula acima!!!*).

(23) (Triplas Pitagóricas – força bruta!) Um triângulo retângulo pode ter lados que são todos inteiros. O conjunto de três inteiros que representam os lados de um triângulo retângulo é chamado de tripla pitagórica. Estes três lados devem satisfazer a relação de que a soma dos quadrados de dois dos lados é igual ao quadrado da hipotenusa. Crie um programa que ache todas as triplas pitagóricas para **lado1**, **lado2** e **hipotenusa**, todos menores do que 500. **Use um laço for triplamente aninhado** que testa todas as possibilidades. Este é um exemplo de calcular pela "força bruta" (você aprenderá, ao longo do curso de Estatística, que existem muitos problemas interessantes para os quais a única solução é testar todas as possibilidades!)