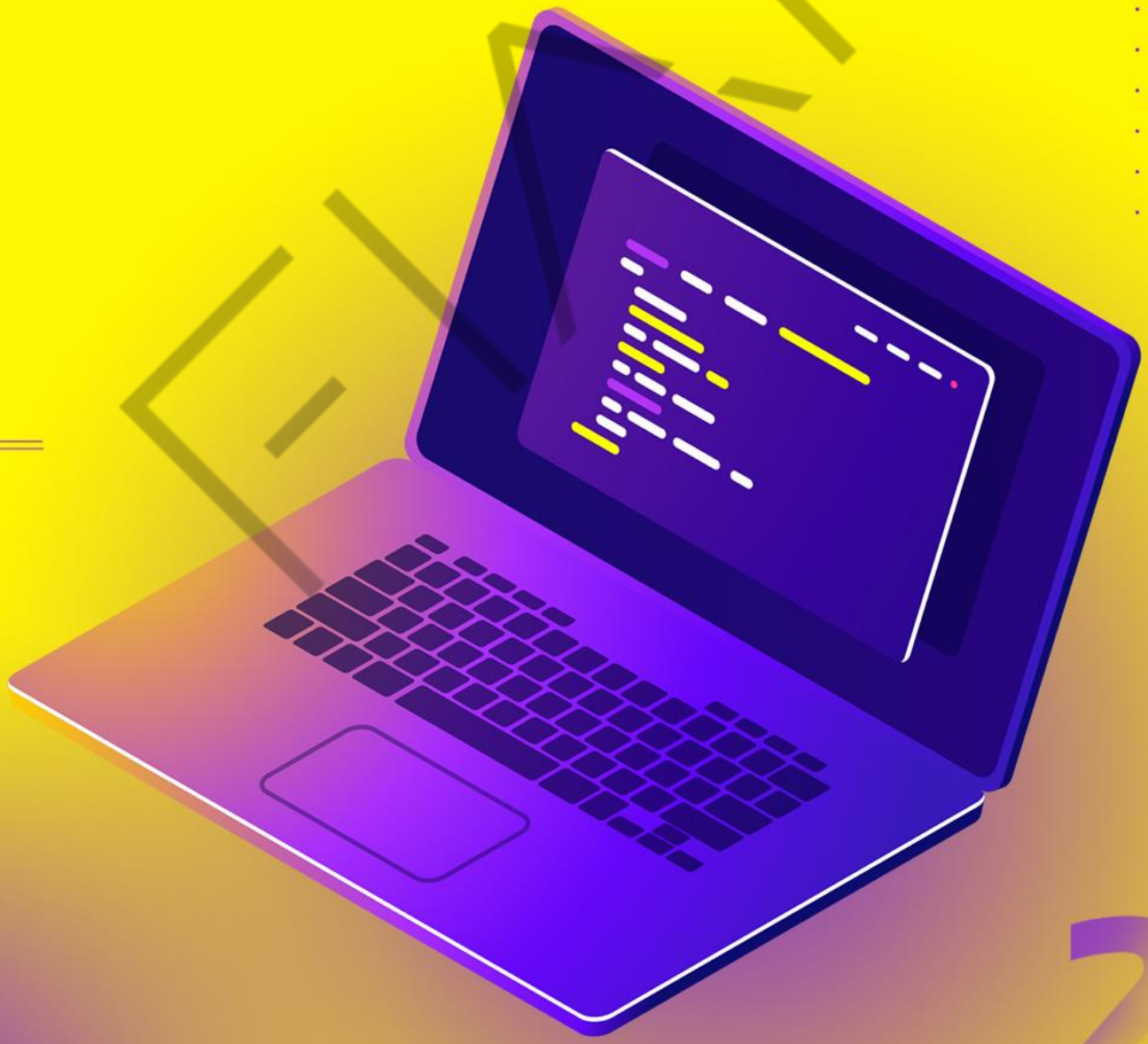
PYTHON

VARIÁVEIS, TOMADA DE DECISÃO *e Laços de Repetição*

HUMBERTO DELGADO DE SOUSA



2

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

2 VARIÁVEIS, TOMADA DE DECISÃO E LAÇOS DE REPETIÇÃO

2.1 Introdução

Antes de iniciarmos o nosso assunto sobre variáveis, vamos primeiramente abrir o nosso projeto criado no PyCharm, no capítulo (MeusProjetos_Python). Para isso, abra o PyCharm e logo surgirá a seguinte caixa de mensagem:

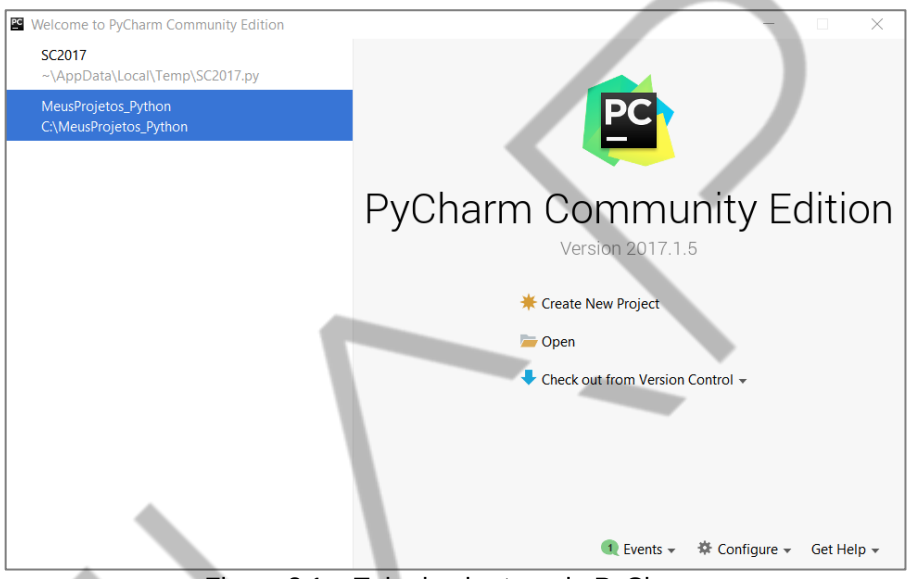


Figura 2.1 – Tela de abertura do PyCharm
Fonte: Elaborado pelo autor (2017)

Clique sobre o seu projeto, conforme destacado em azul na imagem acima, e deverá aparecer a janela abaixo com o código que criamos no final do capítulo anterior, demonstrando, assim, que o PyCharm está pronto para ser utilizado.

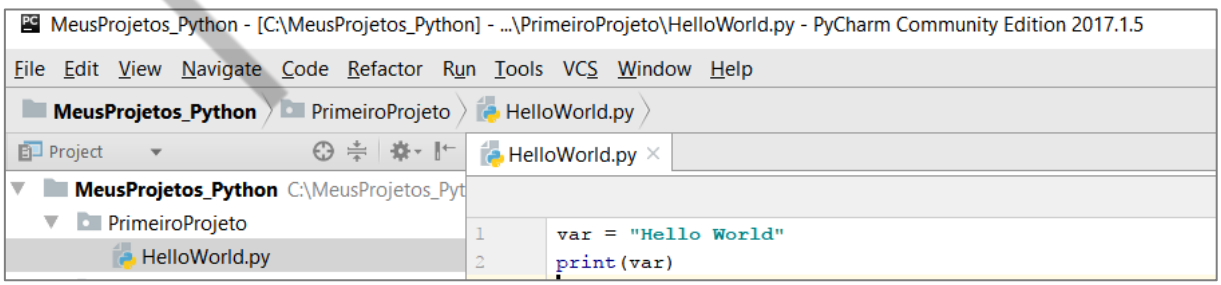


Figura 2.2 – PyCharm com o projeto aberto
Fonte: Elaborado pelo autor (2017)

2.2 O que são variáveis

Variáveis são espaços reservados na memória do computador para que possamos armazenar, temporariamente, um determinado dado.

Vale ressaltar que “dado” é algo que por si só e de maneira isolada não possui significado para o usuário final. O “dado” apenas passa a ter significado para o usuário final quando o combinamos com outros “dados”, tornando-o uma “informação”.

Ficou complicado? Vamos para um exemplo:

Se eu falar para você: dezenove. O que significa? Você pode pensar: é uma dezena ou é uma idade, é a temperatura ambiente ou é a quantidade de alunos, é a quantidade de tarefas ou é a quantidade de pontos que uma equipe de basquete realizou, enfim, o dado “dezenove” por si só não tem significado, pois pode ser qualquer uma dessas hipóteses ou até mesmo todas as hipóteses.

Agora, se eu falar: dezenove sobrinhos, você saberá que estou falando em quantidade de pessoas, mas ainda não tem a informação exata, por quê? Porque faltam dados. Quanto mais dados tivermos, melhor poderá ser a informação que obteremos. Podemos chegar a uma informação do tipo: tenho dezenove sobrinhos, dez com mais de 20 anos e os outros nove com menos ou exatamente 20 anos.

Voltando à definição que iniciamos neste tópico, quando afirmamos que as variáveis armazenam um dado “temporariamente”, isso significa que o dado somente será reconhecido enquanto não for substituído por outro dado, ou ainda, enquanto o seu programa não for encerrado.

Por que, então, devemos criar dados temporários? Não parece ser útil possuirmos dados somente de maneira temporária, certo? Errado! Pense na seguinte situação: quando você entra em um site de compras e coloca alguns itens no carrinho de compras, você pode adicionar ou remover mais itens à vontade, certo? Isso porque você está manipulando dados temporários, que são extremamente velozes pelo fato de estarem localizados dentro da memória RAM do seu computador.

Agora, imagine se, a cada item que você adicionar no carrinho de compras, o site passa a gravar esses dados em um banco de dados ou em um arquivo

qualquer. Toda vez que você remover ou alterar a quantidade dos itens que deseja adquirir, o site deve localizar o dado no banco/arquivo e alterá-lo/removê-lo, consumindo muito mais tempo e recurso, não acha? Vamos esclarecer um pouco mais.

Pense neste exemplo: você chega ao caixa de supermercado com suas compras, aproximadamente 50 itens, se não tivéssemos as variáveis, o caixa do supermercado deveria passar um item, te cobrar, dar o troco, finalizar a venda; passar outro item, te cobrar, dar o troco, finalizar a venda; passar outro item, te cobrar, dar o troco, finalizar a venda; e assim sucessivamente até que tenha feito isso para todos os itens.

Entretanto, como temos as variáveis, isso não ocorre, o caixa do supermercado passa todos os itens, que são armazenados em variáveis (ou em estruturas mais completas que discutiremos posteriormente). Depois, ele cobra o valor, te fornece o troco, se for o caso, e, então, finaliza a venda. Somente nesse momento, todos os dados que estiverem nas variáveis serão armazenados fisicamente em um banco de dados e/ou arquivo. Por isso, grave bem, dados temporários são voláteis e representados, a princípio, por variáveis.

2.3 Identificadores

Como poderemos encontrar um dado dentro de uma variável que, por sua vez, está dentro da memória do computador?

Vamos pensar na prática. Quando você chega em um laboratório de exames clínicos muito movimentado (cada pessoa é uma variável), pega uma senha na entrada e aguarda a chamada. A atendente precisa saber quem é o próximo a ser atendido (dado). E como ela faz isso? Chamando pelos números que ficaram disponíveis na entrada para os clientes, ou seja, ela chama pela sua identificação, o número é o seu identificador.

O mesmo processo vai ocorrer com as variáveis, se eu preciso de um dado que está dentro da variável vou chamar pelo identificador. Com isso, podemos dizer que o identificador é o nome que atribuímos a uma variável. E assim como no mundo real, temos algumas regras e padrões para definirmos um identificador de variável.

Regras: os três itens abaixo são obrigatórios e, caso não sejam seguidos, tornarão o seu código inoperante e a sua aplicação não irá funcionar.

- O nome da variável não pode iniciar com um número. Exemplo: 1berto não pode ser um identificador de variável.
- O identificador não pode fazer uso de palavras reservadas da linguagem. As palavras reservadas são comandos/indicações que pertencem à linguagem. Por isso, não podem ser utilizadas como nomes de variáveis. Por exemplo: *if*, *while*, *for*, entre tantas outras, são palavras reservadas da linguagem Python, não podem ser utilizadas como identificadores.
- Cuidado na utilização de caracteres especiais como: @ # ! ? ( ) [ ] { }, entre tantos outros que não podem ser utilizados em nomes de variáveis, porque também fazem parte da linguagem de programação. O único caractere especial que pode ser utilizado para o identificador de uma variável é o *underline* _.

Padrões: não são obrigatórios, mas são importantíssimos para que possamos manter um código “clean” onde qualquer programador terá facilidade em realizar a leitura do seu código e eventuais manutenções. Utilizar padrões é fazer uso de uma boa prática de programação. Vamos aos padrões:

- O identificador de uma variável sempre deverá começar com letra minúscula (caixa baixa), nunca inicie o nome de uma variável com letra maiúscula.
- Letras maiúsculas somente devem ser usadas quando você fizer uso de duas ou mais palavras para compor o identificador de uma variável. A letra maiúscula identifica o início de uma nova palavra, o que facilita muito para outras pessoas que não conhecem o idioma do código que estiver sendo lido/interpretado. Exemplos válidos: dataDeNascimento ou dataNascimento ou data_nascimento. Como você pode perceber, também pode utilizar o underline para separar as palavras, dispensando o uso dos caracteres maiúsculos no nome das variáveis.
- Utilize nomes significativos, nada de nomes como: xpto, x, y, z, abc, entre tantos outros bem criativos, porém, sem qualquer significado para outros

profissionais. Lembre-se que você irá precisar do nome da sua variável no decorrer do código.

- Acrônimos (siglas que resumem palavras a fim de agilizar o processo de comunicação) devem ser utilizados também com letras minúsculas. Exemplos: cpf, cep, rg, cnpj, nasa.

Vamos verificar alguns identificadores válidos:

numeroElemento2	produto_alimenticio	endereco
código_do_cliente	descricaoCurso	qtdeDeItens

2.4 Tipos de dados

Agora que já definimos o que são variáveis, para que servem e como devemos chamá-las, precisamos definir o tipo do dado que será armazenado dentro da variável. A princípio, dentro da nossa proposta de programação, podemos identificar dois grupos básicos de tipos de dados: numéricos e alfanuméricos.

2.4.1 Tipo numérico

O grupo dos numéricos envolvem os dados que podem participar de alguma operação matemática no decorrer do código, por exemplo: qtdeHorasTrabalhadas, é praticamente certo que o dado desta variável será utilizado a fim de calcular o salário de um funcionário; ou ainda, notaAvaliacao, também outro conteúdo numérico que provavelmente será utilizado para a operação de cálculo da média aritmética.

Já, por exemplo, o numeroTelefone guardaria um dado que não será utilizado para cálculo matemático. Ou você está pensando em dar um desconto sobre o número do seu telefone e não sobre o total da fatura? Nesse caso, o número do telefone, mesmo contendo apenas números, poderá ser definido como alfanumérico. Dentro do grupo numérico, podemos ainda encontrar uma divisão:

- **Números flutuantes:** são os dados que podem fazer uso das casas decimais para uma melhor exatidão, como: salário, valor do litro do combustível, nota de uma prova, altura, peso, entre outros. Esses dados

normalmente são identificados, entre as linguagens de programação, pelos tipos *float* e *double*. O tipo *double* tem o dobro de precisão nas casas decimais, quando comparado com o *float*, por isso, ocupa um espaço maior na memória para alocar o seu dado.

- **Números inteiros:** são os dados que não possuem qualquer necessidade de precisão nas casas decimais. Exemplos: quantidade de faltas, público de um show, quantidade de integrantes de um grupo, quantidade de filhos, quantidade de veículos, entre outros. Os números inteiros são designados nas linguagens como *int* ou *integer*. Existem alguns tipos com capacidades maiores ou menores que o int, mas não serão abordados neste momento, pois fugiria da nossa proposta inicial em usar o Python como uma ferramenta para auxiliar o desenvolvimento de rotinas de automação em um ambiente de infraestrutura.

Para a nossa necessidade, cabe saber que “int” representa números inteiros e “float” representa números decimais, fácil, não é mesmo?

Bem-vindo ao Python!!!!

2.4.2 Tipos alfanuméricos

Consequentemente, o tipo alfanumérico é para aquele dado que você jamais vai utilizar para uma operação matemática qualquer, uma vez que esse tipo é bem menos performático quando comparado aos numéricos. Exemplos de situações em que utilizaríamos os tipos alfanuméricos: nome, rua, número da casa, cep, rg, cargo, entre tantos outros. Tecnicamente, este tipo de dado é conhecido como *string*, pois representa um conjunto de caracteres. Algumas linguagens fazem uso do tipo *char*, que representa apenas um caractere.

Para as nossas aplicações, vamos nos concentrar apenas no tipo “string”, esse será o dado alfanumérico que utilizaremos.

2.5 Criando nossas variáveis

Eis que chega a hora de utilizarmos o Python para colocarmos em prática o que foi visto até aqui. Vamos lá!

Dentro do PyCharm, no projeto “MeusProjetos_Python”, crie uma pasta chamada Capitulo2-Variaveis e, dentro dela, um arquivo chamado: Variaveis.py.

Para quem não se lembra de como fazer isso, clique com o botão direito sobre “MeusProjetos_Python” e siga os menus, conforme imagem abaixo:

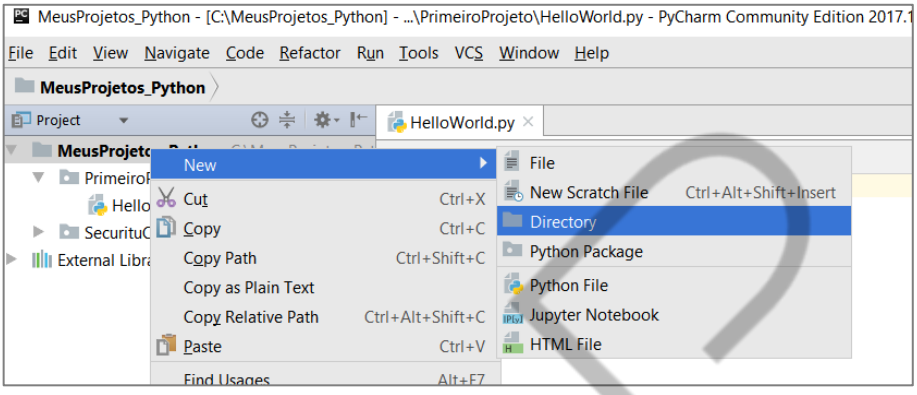


Figura 2.3 – Criação de um diretório dentro de um projeto PyCharm
Fonte: Elaborado pelo autor (2017)

E para criar o arquivo, clique com o botão direito sobre a pasta: “Capitulo2-Variaveis” e selecione as opções, de acordo com a imagem abaixo:

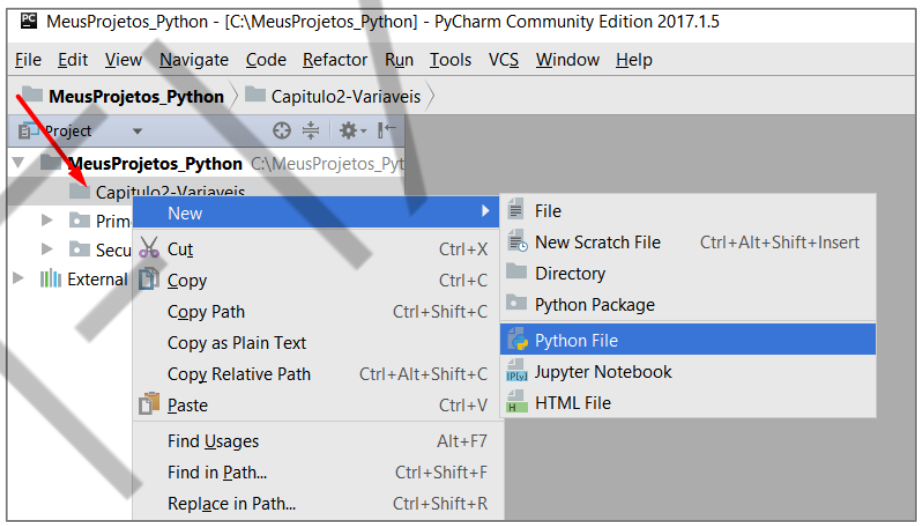


Figura 2.4 – Criação de um arquivo dentro de um diretório no PyCharm
Fonte: Elaborado pelo autor (2017)

Dentro do arquivo Variaveis.py, digite as linhas:

```
nome="Humberto Delgado"
empresa='FIAP'
qtde_funcionarios=500
mediaMensalidade=856.50
```

Código-fonte 2.1 – Criação de variáveis
Fonte: Elaborado pelo autor (2017)

Se você executar o programa, não vai acontecer nada visualmente, pois você está apenas criando as variáveis e atribuindo dados a elas. Repare no código alguns detalhes que podemos destacar:

- Nas linhas 1 e 2, criamos variáveis do tipo String, ou seja, o conteúdo de uma variável String deve sempre estar entre aspas/aspas duplas (“) ou entre apóstrofos/aspas simples (‘).
- Na linha 3, criamos uma variável do tipo *int*.
- Na linha 4, criamos uma variável do tipo *float*. O caractere utilizado para casas decimais é o ponto (.), seguindo o padrão americano, ou seja, vírgula separa casas de milhar e ponto separa casas decimais.

Para que possamos exibir o conteúdo de cada variável, utilizaremos a função ***print()***, por isso, devemos acrescentar ao código as seguintes linhas:

```
nome="Humberto Delgado"
empresa='FIAP'
qtde_funcionarios=500
mediaMensalidade=856.50
print(nome + " trabalha na empresa " + empresa)
print("Possui: ", qtde_funcionarios, " funcionarios.")
print("A média da mensalidade é de: " + str(mediaMensalidade))
```

Código-fonte 2.2 – Exibição de variáveis
Fonte: Elaborado pelo autor (2017)

Detalhando:

- Na linha 5, exibimos o valor das duas primeiras variáveis, separadas por um texto; para unir o conteúdo delas, utilizamos o operador “+”, que representa concatenação/junção para string´s.
- Na linha 6, onde estamos exibindo a variável inteira e dois textos, utilizamos a “virgula” para indicar o final do primeiro texto, a variável e o início do segundo texto, sem precisar converter a variável numérica para string.
- Na linha 7, exibimos um texto e uma variável do tipo “float”, concatenadas através do operador “+”. Para que o Python não tente realizar uma operação matemática, devemos utilizar a função ***str()*** para converter o conteúdo da variável “float” para string, por isso a variável mediaMensalidade está dentro da função ***str()***.

Podemos exibir as variáveis numéricas (int ou float), com as string´s, utilizando a forma definida na linha 6 ou conforme apresentado na linha 7. Execute e verá a saída dos dados das variáveis transformada em informação para o usuário final. Caso não lembre como executar a sua aplicação, clique no seu arquivo "Variaveis" com o botão direito e selecione a opção "Run Variaveis".

Entretanto, se você executar dez vezes o seu arquivo, verá a mesma mensagem, ou seja, suas variáveis não estão variando tanto assim, não é mesmo? Por isso, vamos utilizar a função ***input()***, assim como a função *print*() exibe. A função input() permitirá que o usuário final possa digitar um conteúdo em tempo de execução.

Vamos realizar a seguinte alteração no código:

```
nome=input("Digite um funcionário: ")
empresa=input("Digite a instituição: ")
qtde_funcionarios=int(input("Digite a qtde de funcionários: "))
mediaMensalidade=float(input("Digite a média da mensalidade: "))
print(nome + " trabalha na empresa " + empresa)
print("Possui: ", qtde_funcionarios, " funcionarios.")
print("A média da mensalidade é de: " + str(mediaMensalidade))
```

Código-fonte 2.3 – Utilização do input()
Fonte: Elaborado pelo autor (2017)

Execute e veja que as variáveis agora realmente estão funcionando como variáveis, pois, a cada execução, os valores podem mudar. Atente-se que você deverá digitar os valores na parte inferior da tela, conforme a imagem abaixo:

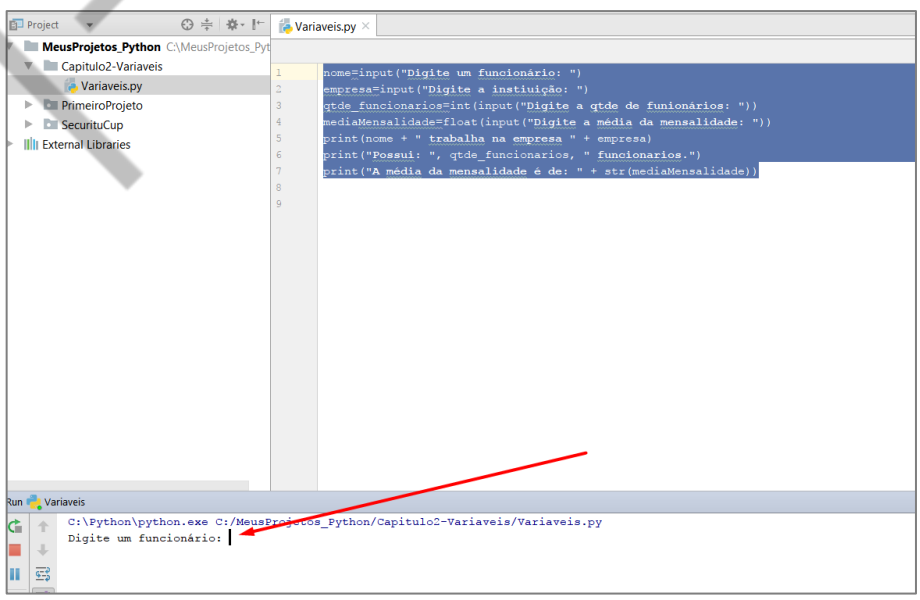


Figura 2.5 – Preenchimento de um *input*()
Fonte: Elaborado pelo autor (2017)

Detalhe para as linhas 3 e 4 agora, onde estamos fazendo conversões de *string* para *int* (na linha 3) e de *string* para *float* (na linha 4). Isso porque o *input*() captura apenas *strings*, logo após capturado o valor, convertemos para o tipo de dado que desejamos armazenar em sua respectiva variável.

Para finalizar, vamos acrescentar mais quatro linhas no final do código e poderemos comprovar os tipos de dados que estão sendo utilizados pelas variáveis. Utilizaremos mais uma função, chamada ***type()***. Esta função retorna o tipo do dado do que estiver dentro dos seus parênteses, conforme observamos no código abaixo:

```
nome=input("Digite um funcionário: ")
empresa=input("Digite a instituição: ")
qtde_funcionarios=int(input("Digite a qtde de funcionários: "))
mediaMensalidade=float(input("Digite a média da mensalidade: "))
print(nome + " trabalha na empresa " + empresa)
print("Possui: ", qtde_funcionarios, " funcionarios.")
print("A média da mensalidade é de: " + str(mediaMensalidade))
print("==============Verifique os tipos de dados abaixo:==============")
print("O tipo de dado da variavel [nome] é: ",type(nome))
print("O tipo de dado da variavel [empresa] é: ",type(empresa))
print("O tipo de dado da variavel [qtde_funcionarios] é:
",type(qtde_funcionarios))
print("O tipo de dado da variavel [mediaMensalidade] é:
",type(mediaMensalidade))
```

Código-fonte 2.4 – Utilização do type()
Fonte: Elaborado pelo autor (2017)

Para encerrar esta parte, pense no texto abaixo:

“Declaro para o senhor Gonçalves Dias que o senhor Humberto Delgado esteve presente no evento SecurityCup e gastou o valor de R$ 30,00 com a entrada.”

Identifique o que pode ser variável no texto acima e monte um projeto no Python, em um novo arquivo chamado Variaveis2. No final, deverá exibir o texto completo com o valor das variáveis. Agora, pare, siga para o PyCharm e tente resolver!

Bem, o seu resultado deve estar semelhante ao código abaixo:

```
responsavel=input("Digite o nome do responsável: ")
funcionario=input("Digite o nome do funcionário: ")
evento=input("Digite o nome do evento: ")
valor=float(input("Digite o valor que será ressarcido: "))

print("Declaro para o senhor " + responsavel + ", que o senhor " +
funcionario + " esteve presente no evento " + evento + " e gastou o valor
de R$ " + str(valor) + " com a entrada.")
```

Código-fonte 2.5 – Utilização do type()
Fonte: Elaborado pelo autor (2017)

Cuidado apenas com o valor, que não é *string,* por isso, precisa ser convertido dentro da função **print()**!

2.6 Tomadas de decisões

Durante o desenvolvimento de um código, existem diversas situações em que a resolução dependerá de um valor para que possa seguir um rumo e, consequentemente, que uma tomada de decisão também seja seguida. Isso ocorre muito em nosso dia a dia quando planejamos uma tarefa, mas as “variáveis” do nosso cotidiano fazem com que tenhamos que optar por mudanças de planos. Um exemplo típico é quando estimamos um determinado tempo para chegarmos a um local e um simples pneu furado do seu veículo atrapalha todo o planejamento do tempo outrora estimado.

Para que estas tomadas de decisões sejam possíveis, utilizaremos um comando condicional muito conhecido em todas as linguagens de programação, identificado como “*if*”. Este comando será responsável por acrescentarmos ao nosso bloco de código a inteligência da tomada de decisão, tornando-o mais independente.

A forma de utilizar o comando “*if*”, ou seja, sua sintaxe, é bem simples:

if <condição>:

<o que você quer que aconteça caso a condição seja verdadeira>

Repare em dois detalhes importantes:

- A linha do *if* deve ser encerrada com dois pontos (:).
- A(s) linha(s) a ser executada(s), caso a condição seja verdadeira, deverá(ão) estar com um recuo da margem esquerda. Este recuo deve ser realizado através da tecla <TAB>. Tudo que estiver recuado, abaixo do *if*, será executado somente se a condição for verdadeira.

Exemplo: veja, no código abaixo, que somente as linhas apontadas com a seta vermelha estão com recuo, portanto, somente elas são dependentes da

condição. Se a nota for maior ou igual a seis, elas serão executadas, caso contrário, não serão executadas.

```
nota=8
if nota>=6:
    print("Você está de...")
    print("PARABÉNS!!!")
    print("Este texto ainda será exibido se a nota for maior que seis")
print("A exibicao deste texto independe da condição do if")
```

Código-fonte 2.6 – Uso do comando "if"
Fonte: Elaborado pelo autor (2017)

Podemos utilizar o comando *"if"* em três estruturas distintas, como observamos a seguir.

2.6.1 Decisões simples

Uma decisão simples é o caso mais comum e corriqueiro que existirá dentro de um código. Imagine que você precisará coletar o nome dos pacientes que serão atendidos em uma sala de emergência em um hospital, porém algumas das pessoas deverão ter prioridade no atendimento, portanto, vamos definir um atendimento preferencial para os idosos, que podem ser identificados pela idade maior ou igual a 65 anos. Nosso código, então, precisará saber o nome e a idade da pessoa, para que possamos definir se terá atendimento prioritário ou não.

No PyCharm, crie uma pasta chamada "Capitulo2-Decisoes" (evite utilizar acentos e cedilha para nomes de arquivos e diretórios), dentro dela, crie um arquivo Python chamado "DecisaoSimples.py" e digite o código abaixo:

```
nome=input("Digite o nome: ")
idade=int(input("Digite a idade: "))
prioridade="NÃO"
if idade>=65:
    prioridade="SIM"
print("O paciente " + nome + " possui atendimento prioritário? " + prioridade)
```

Código-fonte 2.7 – Exemplo de uma tomada de decisão simples
Fonte: Elaborado pelo autor (2017)

Veja que acima o código irá mudar o valor da variável "prioridade" de acordo com o valor da idade. Percebeu como o código vai ficando mais independente? Ele

só precisa dos dados para que possa retornar uma informação. A última linha (print) será executada independentemente da idade digitada.

2.6.2 Decisões compostas

As tomadas de decisões compostas, por sua vez, são formadas para o direcionamento caso a condição seja verdadeira e caso a condição seja falsa, ou seja, ela deve avaliar as duas hipóteses. Por isso, a nossa estrutura do comando if vai alterar um pouco, entraremos com um comando chamado “else”, que executará as linhas tabuladas dentro dele, caso a condição seja falsa.

Vamos criar um arquivo chamado “DecisaoComposta” (dentro da pasta Capitulo2-Decisoes) e reproduzir o código abaixo:

```
nome=input("Digite o nome: ")
idade=int(input("Digite a idade: "))
if idade>=65:
    print("O paciente " + nome + " POSSUI atendimento prioritário!")
else:
    print("O paciente " + nome + " NÃO possui atendimento prioritário!")
```

Código-fonte 2.8 – Exemplo de uma tomada de decisão composta
Fonte: Elaborado pelo autor (2017)

Agora economizamos recursos. Perceba que, no exemplo anterior, (DecisaoSimples), utilizamos três variáveis e, como já mencionado, variáveis ocupam espaço em memória. Neste exemplo, usamos apenas duas variáveis e a informação está chegando ao usuário final da mesma forma, ou seja, utilizamos menos espaço da memória, e isto é muito bom. Preste muita atenção nos recuos, perceba que o “else:” está alinhado com o “if” e que cada comando possui uma linha tabulada. Por isso, podemos afirmar categoricamente que somente um print() será executado, vai depender somente da idade do paciente.

Alguns programadores menos avisados utilizam duas tomadas de decisões simples em vez de uma composta, **NUNCA** faça isso. Veja como ficaria o código:

```
nome=input("Digite o nome: ")
idade=int(input("Digite a idade: "))
if idade>=65:
    print("O paciente " + nome + " POSSUI atendimento prioritário!")
if idade < 65:
    print("O paciente " + nome + " NÃO possui atendimento prioritário!")
```

Código-fonte 2.9 – Exemplo de má prática de programação com “ifs”
Fonte: Elaborado pelo autor (2017)

O código acima está totalmente fora das boas práticas de programação, os dois “ifs” serão processados sempre que o programa for executado. Observe o seguinte: se o paciente tiver 80 anos, o primeiro “if” será processado, mesmo assim, a linguagem verificará o segundo “if”, e sabemos que não existe chance alguma de a condição do segundo “if” ser verdadeira, ou seja, a linguagem vai consumir um processamento desnecessário.

Se você avaliar que a condição que estamos propondo (uma comparação com números) é uma condição que não exige muito esforço do processador, você não verá problemas, mas pense que futuramente não trabalharemos apenas com condições tão simples e, com certeza, vamos exigir esforço do processador, por isso, devemos seguir as boas práticas desde já. NÃO substitua NUNCA uma decisão composta por duas ou mais decisões simples.

Também poderemos encontrar uma situação em que o Verdadeiro e o Falso, simplesmente, não sejam suficientes. Por exemplo, vamos imaginar que pessoas com idade igual ou superior a 65 receberão atendimento prioritário, mas que também pessoas com suspeita de doenças infecto-contagiosas deverão ser direcionadas para uma sala de espera distinta, por motivos óbvios.

Este é um caso em que um Verdadeiro ou um Falso na idade não resolve, pois precisamos verificar se a idade do paciente é maior ou igual a 65 anos e, se a resposta for Falsa, devemos verificar ainda se o paciente está com suspeita de doença infecto-contagiosa. E se, ainda assim, for Falsa a condição, então, podemos considerar que o paciente deve aguardar sem prioridade na sala comum de espera. Vamos alterar o nosso código proposto anteriormente para este novo cenário, da seguinte forma:

```
nome=input("Digite o nome: ")
idade=int(input("Digite a idade: "))
doenca_infectocontagiosa=input("Suspeita de doença infecto-
contagiosa?").upper()
if idade>=65:
    print("O paciente " + nome + " POSSUI atendimento prioritário!")
elif doenca_infectocontagiosa=="SIM":
    print("O paciente " + nome + " deve ser direcionado para sala de espera
reservada.")
else:
    print("O paciente " + nome + " NÃO possui atendimento prioritário e
pode aguardar na sala comum!")
```

Código-fonte 2.10 – Exemplo de utilização do comando “elif”
Fonte: Elaborado pelo autor (2017)

Um destaque para o final da terceira linha, na qual utilizamos uma função chamada “upper()”. Esta função tem a finalidade de converter o conteúdo de uma string para letras maiúsculas, ou seja, o que for digitado no “input()”, pelo usuário final, será convertido para caixa alta (letras maiúsculas), assim fica mais fácil realizar a comparação na condição dentro do “if”.

Veja que implementamos no código acima um comando chamado “elif:”, que é uma sigla para “else”+”if”, ou seja, existe uma segunda condição que deve ser também avaliada. Podemos acrescentar quantos “elifs” forem necessários.

Repare também ainda na linha do “elif:” que utilizamos o operador “==”, o qual deve ser implementado para indicar comparação, por isso, muito CUIDADO, **um “=” representa atribuição, dois “==” representam comparação**.

Dessa forma, o “else” somente será executado se a primeira e a segunda condição forem falsas.

Mas e se o paciente tiver idade igual ou maior a 65 anos e também possuir suspeita de doença infecto-contagiosa? Fez esse teste? Tente! O paciente terá atendimento prioritário, MAS não será direcionado para a sala de espera reservada. Problemas à vista... solução?!?!

Vamos utilizar os operadores “AND” ou “OR”, que nos permitem colocar duas ou mais condições dentro de um único “if” e/ou “elif”. Quando utilizamos o operador “AND”, queremos dizer que a condição que estiver à esquerda e a condição que estiver à direita do operador devem ser verdadeiras para que o “if” seja considerado Verdadeiro. Se uma das condições retornarem Falso, o “if” irá retornar Falso.

Já o operador “OR” determina que se uma das condições for verdadeira, o “if” já deverá retornar “Verdadeiro”. Para simplificar o que queremos fazer, vamos imaginar a seguinte situação: o paciente chega à triagem que o direcionará à sala de espera da consulta. O funcionário da triagem poderá direcionar o paciente para uma das duas salas de espera, são elas:

- Branca: para os pacientes sem risco de doença infecto-contagiosa, mas com ou sem prioridade.

- Amarela: para os pacientes com risco de doença infecto-contagiosa, mas com ou sem prioridade.

Vamos alterar o nosso código para a seguinte forma:

```
nome=input("Digite o nome: ")
idade=int(input("Digite a idade: "))
doenca_infectocontagiosa=input("Suspeita de doença infecto-
contagiosa?").upper()
if idade>=65 and doenca_infectocontagiosa=="SIM":
    print("O paciente será direcionado para sala AMARELA - COM prioridade")
elif idade < 65 and doenca_infectocontagiosa == "SIM":
    print("O paciente será direcionado para sala AMARELA - SEM prioridade")
elif idade >= 65 and doenca_infectocontagiosa == "NAO":
    print("O paciente será direcionado para sala BRANCA - COM prioridade")
elif idade < 65 and doenca_infectocontagiosa == "NAO":
    print("O paciente será direcionado para sala BRANCA - SEM prioridade")
else:
    print("Responda a suspeita de doença infectocontagiosa com SIM ou NAO")
```

Código-fonte 2.11 – Exemplo de utilização do operador “and”
Fonte: Elaborado pelo autor (2017)

Da forma como está o código acima, fechamos todas as possibilidades, e o “else:” será executado somente se o usuário final digitar algo diferente de “SIM” ou “NAO”, uma vez que a idade, pelo fato de ser numérica, está cercada e não há previsão de uma idade inválida.

2.6.3 Decisões encadeadas

As decisões encadeadas servem para avaliarmos uma situação e, dependendo da situação, serão tomadas algumas outras decisões. Por isso, servem como uma alternativa para estruturas mais robustas, como a que foi vista no último tópico ou ainda para verificar mais possibilidades para um mesmo dado sem a utilização dos operadores “and” ou “or”.

Podemos pensar em uma string que não tenha apenas os valores “SIM” e “NAO”, como o estado civil, que pode ter mais do que simplesmente dois valores (solteiro, casado, divorciado, viúvo etc.). Pode ser ainda que precisaremos avaliar cada valor individualmente, ou grupos de situações, tornando o “and” ou o “or” mais confuso e/ou mais complexo na sua implementação.

Crie um arquivo no diretório “Capitulo2-Decisoes” chamado “DecisaoEncadeada.py” e veja como ficaria o mesmo código do último tópico, mas agora com decisão encadeada.

```
nome=input("Digite o nome: ")
idade=int(input("Digite a idade: "))
doenca_infectocontagiosa=input("Suspeita de doença
infectocontagiosa?").upper()
if idade >= 65:
    print("Paciente COM prioridade")
    if doenca_infectocontagiosa=="SIM":
        print("Encaminhe o paciente para sala AMARELA")
    elif doenca_infectocontagiosa=="NAO":
        print("Encaminhe o paciente para sala BRANCA")
    else:
        print("Responda a suspeita de doença infectocontagiosa com SIM ou
NAO")
else:
    print("Paciente SEM prioridade")
    if doenca_infectocontagiosa=="SIM":
        print("Encaminhe o paciente para sala AMARELA")
    elif doenca_infectocontagiosa=="NAO":
        print("Encaminhe o paciente para sala BRANCA")
    else:
        print("Responda a suspeita de doença infectocontagiosa com SIM ou
NAO")
```

Código-fonte 2.12 – Exemplo de utilização com decisão encadeada
Fonte: Elaborada pelo autor (2017)

O resultado final para o usuário final é o mesmo, a diferença é que iremos resolvendo o problema por partes. Levando em consideração o código acima, primeiro vamos identificar a idade do paciente para definir se ele terá atendimento prioritário ou não, e depois se o paciente está ou não sob a situação suspeita de doença infecto-contagiosa.

Você deve estar se perguntando: então, tanto faz usar decisão composta ou decisão encadeada? Para o exemplo acima, poderíamos dizer que sim, mas, na prática, não é bem assim. Observe que temos apenas duas condições (idade e suspeita de doença infecto-contagiosa). Se tivéssemos mais condições, teríamos uma sequência muito grande de “if´s” e “elifs” e, consequentemente, uma grande chance de errarmos uma das linhas de condições compostas. Neste caso, é altamente recomendável, por boa prática, utilizarmos as decisões encadeadas, pois facilitam a leitura e isolam as ações que devem ser levadas em consideração.

Além disso, a decisão encadeada foi desenvolvida para que se possa tomar uma decisão somente quando outra decisão já foi definida como verdadeira ou falsa.

Vamos para o seguinte caso: mulheres grávidas também são consideradas para o atendimento prioritário (sala Branca ou Amarela). Você vai perguntar para todos os pacientes se eles estão grávidos? Não, apenas para as mulheres. Então,

você perguntaria para todas as mulheres? Não, você não precisaria perguntar para as mulheres com idade igual ou superior a 65 anos, assim também poderia descartar crianças com menos de 10 anos.

Logo, não seria possível resolver essa situação, de maneira lógica e prática, utilizando a decisão composta. Vamos criar outro arquivo chamado: “DecisaoEncadeada2.py” e montaremos o seguinte código:

```
nome=input("Digite o nome: ")
idade=int(input("Digite a idade: "))
doenca_infectocontagiosa=input("Suspeita de doença infectocontagiosa?
").upper()

# PRIMEIRO PROBLEMA A SER RESOLVIDO
if doenca_infectocontagiosa=="SIM":
    print("Encaminhe o paciente para sala AMARELA")
elif doenca_infectocontagiosa=="NAO":
    print("Encaminhe o paciente para sala BRANCA")
else:
    print("Responda a suspeita de doença infectocontagiosa com SIM ou NAO")

# SEGUNDO PROBLEMA A SER RESOLVIDO
if idade >= 65:
    print("Paciente COM prioridade")
else:
    genero=input("Digite o gênero do paciente: ").upper()
    if genero=="FEMININO" and idade>10:
        gravidez=input("A paciente está grávida? ").upper()
        if gravidez=="SIM":
            print("Paciente COM prioridade")
        else:
            print("Paciente SEM prioridade")
    else:
        print("Paciente SEM prioridade")
```

Código-fonte 2.13 – Exemplo 2 de utilização com decisão encadeada
Fonte: Elaborado pelo autor (2017)

Uauuu... perceba agora como o nosso código está legível e claramente com dois problemas sendo resolvidos.

Delimitamos os dois problemas utilizando o caractere “#’. Este caractere possui muitos nomes: *hashtag*, o mais descolado; mas também pode ser: jogo da velha, sustenido, cerquilha, antífen, quadrado e o mais desejado de todos os programadores: lasanha, porque ele parece uma lasanha, não é mesmo? Enfim, sua função é a de indicar que a linha iniciada com este caractere não deve ser compilada, ou seja, não é uma linha de comandos e, sim, uma linha particular de comentários do programador.

Seguindo a avaliação do código, identificamos se o paciente está com suspeita de doença infecto-contagiosa, pois, independentemente do gênero ou da idade, ele será deslocado para uma sala AMARELA, caso a condição seja verdadeira, ou para a sala BRANCA, no caso de falsa. E se a informação não estiver dentro do esperado, o usuário deverá ser informado de que foi impossível definir a sala.

O segundo problema é em relação à prioridade, ou seja, se o paciente terá ou não atendimento prioritário. Identificamos, primeiramente, se a sua idade é igual ou superior a 65 anos. Se isso for verdade, independentemente da condição, ele terá prioridade no atendimento. Se ele tiver menos que 65 anos, ainda não é possível definir que não terá atendimento prioritário, pois pode se tratar de uma mulher grávida.

Logo, perguntaremos para o paciente o seu gênero, se não for feminino, já saberemos que é uma pessoa do sexo masculino com menos de 65 anos, ou seja, não terá prioridade. Mas se for feminino, deve-se perguntar se está grávida, entretanto, essa pergunta deve ser feita apenas para mulheres com mais de 10 anos. Se for mulher e tiver menos que 10 anos, já podemos classificar como não prioritário. Se for mulher com mais de 10 anos e, ao ser perguntada se está grávida, a resposta for “SIM”, o atendimento será prioritário, caso contrário, continua sem prioridade na consulta.

O mais importante é observar que os recursos são solicitados de acordo com a situação real, não seria legal realizar uma série de perguntas para o usuário final sabendo-se que algumas situações já poderiam ser definidas, sem perguntas/ recursos/dados desnecessários. Isso somente é possível devido ao uso das decisões encadeadas. Desafiador, não é mesmo?

Procure criar um arquivo chamado DesafioDecisao.py e elaborar o código para resolver a seguinte situação: o seu módulo solicitará o nível de acesso de uma pessoa que pode ser: ADM, USR ou GUEST e o gênero dessa pessoa, caso o nível seja ADM, ele deverá exibir “Olá administrador” para os homens ou “Olá administradora” para as mulheres. Se o nível for USR, deverá exibir “Olá usuário” para os homens ou “Olá usuária” para as mulheres. Se o nível for GUEST, a mensagem deverá ser “Olá visitante”. E se o nível digitado for diferente de ADM, USR ou GUEST deverá exibir a mensagem “Olá desconhecido(a)”. Considere

apenas os gêneros masculino e feminino e não olhe o código abaixo, até resolver o seu desafio.

```
nivel=input("Digite o nível de acesso: ").upper()
if nivel=="ADM" or nivel=="USR":
    genero=input("Digite o seu gênero: ").upper()
    if nivel=="ADM":
        if genero=="FEMININO":
            print("Olá administradora")
        else:
            print("Olá administrador")
    else:
        if genero=="FEMININO":
            print("Olá usuária")
        else:
            print("Olá usuário")
elif nivel=="GUEST":
    print("Olá visitante")
else:
    print("Olá desconhecido(a)")
```

Código-fonte 2.14 – Desafio de tomada de decisão
Fonte: Elaborado pelo autor (2017)

Fácil, não é mesmo? Mas tenha sempre muito cuidado com as tabulações, elas são decisivas em Python.

2.7 Laços de repetições

Imagine, sobre o nosso último código digitado, a seguinte situação: deveremos inserir os dados de 10 pacientes no sistema. Para você realizar este cadastro, deveria executar o mesmo programa 10 vezes. Imagine, então, agora, cadastrar 100 pacientes, 1.000 pacientes, o nosso código está pouco produtivo, concorda? É aí que entram as repetições.

Os laços de repetições servem para que uma determinada ação (representada por um conjunto de códigos) seja repetida uma determinada ou indeterminada quantidade de vezes. Esse recurso é muito importante dentro da programação, pois permite uma redução drástica em relação à quantidade de linhas digitadas.

Basicamente, no Python, encontramos dois comandos que podem ser utilizados para realizar repetições de trechos de códigos, são eles: *while* e *for*. Veremos, a seguir, as diferenças entre eles.

2.7.1 While

A estrutura “while” é a mais comum entre os programadores, e normalmente a primeira que se aprende. Ela funciona basicamente como um “if”, mas com a diferença que executará o seu bloco de códigos enquanto a condição for verdadeira e não somente uma vez, ou seja, a condição será testada enquanto não for falsa. Podemos dizer que o fluxo do código seguirá a seguinte sequência:

- Testa a condição – se verdadeira.
- Executa o bloco de código.
- Volta para condição.

Repare que se a condição for eternamente verdadeira, você criará o que chamamos de “*loop infinito*”, isto é, o programa ficará executando este bloco de código até que o programa seja encerrado de maneira abrupta, seja por falha do sistema, estouro da memória, computador foi desligado ou algo do gênero. Esse deve ser o nosso maior cuidado, em não criarmos “loops infinitos”.

Vamos aos códigos para que possamos verificar na prática os conceitos e o comando que foi apresentado. Crie dentro do seu projeto do PyCharm um novo diretório chamado “Capitulo2-Repeticoes” e, dentro dele, crie um arquivo Python chamado “while_infinito.py” e monte o código abaixo:

```
numero=int(input("Digite um numero: "))
while numero<100:
    print("\t" + str(numero))
    numero=numero+1
print("Laço encerrado....")
```

Código-fonte 2.15 – Exemplo de laço infinito
Fonte: Elaborado pelo autor (2017)

Vamos às novidades do código acima representado:

- Conseguiu verificar que a forma de usar (sintaxe) o “while” é idêntica à do comando “if”?
- Dentro da string que está dentro do print(), utilizamos o caractere “\t” (lemos: contra barra t), cuja função é acrescentar uma tabulação antes do número, um detalhe apenas para que o número não saia impresso colado na margem esquerda do console. Podemos utilizar vários caracteres

especiais dentro de strings. Outro bem comum é o “\n”, que insere uma quebra de linha, um <*enter*> (teste, no código, a substituição do “t” pelo “n”). Existem diversos caracteres especiais que seguiremos aprendendo no decorrer do conteúdo do material.

- Na linha 4, é o que garantimos que o nosso “while” seja finito, pois a condição é continuar desde que o valor da variável “numero” seja menor que 100. Conforme vamos acrescentando “um” ao conteúdo dela a cada repetição, chegará o momento que ela atingirá o valor 100 e, então, encerrará o laço, pois a condição retornará Falsa. Esta linha também poder ser substituída pela linha: numero+=1. O efeito será o mesmo.
- Caso o valor digitado pelo usuário final seja um valor maior ou igual a 100, nada que está dentro do laço será executado, o programa vai pular para a linha 5, que está fora do laço, de acordo com a tabulação.
- Você pode alterar na linha o valor “1” por outro número qualquer, a fim de verificar que o valor da variável “numero” irá saltar de acordo com a alteração que fizer nesta operação de soma (atribuição).
- Experimente comentar a linha 4 (coloque o # no início da linha), você verá que cairá em um “loop infinito”, pois o valor da variável não será alterado em momento algum, ou seja, a condição será eternamente verdadeira. Para encerrar sua aplicação de maneira abrupta sem maiores prejuízos, clique no item destacado na figura abaixo.

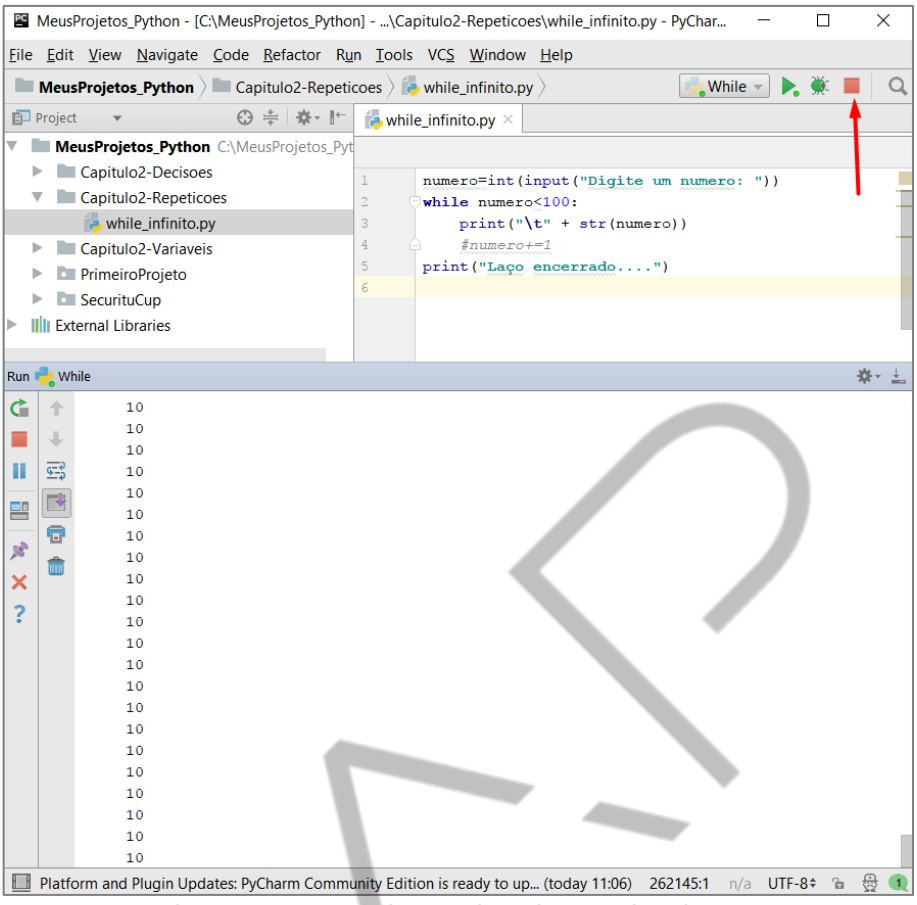


Figura 2.6 – Parando o projeto de maneira abrupta
Fonte: Elaborada pelo autor (2017)

Agora que estamos mais familiarizados com o que é um laço e qual a sintaxe do comando “while”, vamos aplicá-lo a algo mais prático e usual. Crie um arquivo chamado: “DesafioDecisaoWhile.py”, copie o código do arquivo “DesafioDecisao.py” que está na pasta “Capitulo2-Decisoes” e cole dentro do arquivo criado. Insira as linhas que estão na cor vermelha, acrescente um <TAB> na frente de todas as linhas que foram copiadas e deixe o código da seguinte forma:

```
resposta="SIM"
while resposta=="SIM":
    nivel=input("Digite o nível de acesso: ").upper()
    if nivel=="ADM" or nivel=="USR":
        genero=input("Digite o seu gênero: ").upper()
        if nivel=="ADM":
            if genero=="FEMININO":
                print("Olá administradora")
            else:
                print("Olá administrador")
        else:
            if genero=="FEMININO":
                print("Olá usuária")
            else:
                print("Olá usuario")
    elif nivel=="GUEST":
        print("Olá visitante")
```

```
    else:
        print("Olá desconhecido(a)")
    resposta=input("Digite SIM para continuar: ").upper()
```

Código-fonte 2.16 – Desafio com “while”
Fonte: Elaborado pelo autor (2017)

Agora procure fazer o mesmo com o conteúdo do arquivo “DecisaoEncadeada2.py” que está no diretório “Capitulo2-Decisoes”, vamos lá:

- Copie o conteúdo do arquivo “DecisaoEncadeada2.py” que está no diretório “Capitulo2-Decisoes”.
- Crie um novo arquivo na pasta Capitulo2-Repeticoes chamado “DecisaoEncadeada2While.py” e cole dentro deste arquivo o conteúdo que foi copiado.
- Queremos obter o seguinte resultado: enquanto o usuário digitar algo diferente (representamos diferente dentro do Python, através do seguinte operador: !=) de “SIM” ou “NAO”, o programa continuará perguntando se o paciente suspeita ter alguma doença infecto-contagiosa. Tente primeiro fazer sozinho, sem olhar o código abaixo.
- O código deve ter ficado semelhante ao apresentado a seguir:

```
nome=input("Digite o nome: ")
idade=int(input("Digite a idade: "))
doenca_infectocontagiosa=input("Suspeita de doença infecto-contagiosa? ").upper()
# PRIMEIRO PROBLEMA A SER RESOLVIDO
while doenca_infectocontagiosa!="SIM" and  doenca_infectocontagiosa!="NAO":
    print("Digite SIM ou NAO")
    doenca_infectocontagiosa = input("Suspeita de doença infecto-contagiosa? ").upper()

if doenca_infectocontagiosa=="SIM":
    print("Encaminhe o paciente para sala AMARELA")
else:
    print("Encaminhe o paciente para sala BRANCA")

# SEGUNDO PROBLEMA A SER RESOLVIDO
if idade >= 65:
    print("Paciente COM prioridade")
else:
    genero=input("Digite o gênero do paciente: ").upper()
    if genero=="FEMININO" and idade>10:
        gravidez=input("A paciente está grávida? ").upper()
        if gravidez=="SIM":
            print("Paciente COM prioridade")
        else:
            print("Paciente SEM prioridade")
    else:
        print("Paciente SEM prioridade")
```

Código-fonte 2.17 – Desafio 2 com “while”
Fonte: Elaborado pelo autor (2017)

Repare que o código alterado ocorreu apenas no “primeiro problema”, e que a estrutura do “if” logo abaixo do “while” também foi alterada. Não faz mais sentido ter o “elif”, pois somente serão aceitos os valores “SIM” e “NAO”. Qualquer valor fora disso ficará preso no “while”, até que se digite o valor desejável.

Dentro do código apresentado, podemos utilizar ainda outro “while” para que permita adicionar mais que um paciente, assim como fizemos com o usuário no segundo exemplo deste tópico. Tente, mãos na massa!

2.7.2 For

A estrutura “for” também permite uma repetição, mas sua aplicação é um tanto quanto diferente quando comparada ao “while”. O “for” indica o término do laço de duas formas básicas: por um número que delimita o seu final ou por uma lista de dados que foi verificada por completo.

Em suma, podemos afirmar que o “while” trabalha mais diretamente com a interação do usuário final, diferentemente do “for”, com o que normalmente a situação está previamente definida pelo próprio sistema. No atual momento, diria para vocês fixarem-se mais no “while”. Logo, com a evolução dos capítulos, vocês constatarão, de maneira mais clara, a diferença entre as aplicações dos dois comandos. De qualquer forma, vamos seguir com alguns exemplos para que você já possa conhecer a estrutura do “for”.

Crie um arquivo chamado “exemplo_for” dentro da pasta “Capitulo2-Repeticoes” e monte o seguinte código:

```
for numero in range(1,int(input("Digite um numero para determinar o fim: ")),1):
    print ("\t" + str(numero))
```

Código-fonte 2.18 – Exemplo com o comando “for”
Fonte: Elaborado pelo autor (2017)

Para o comando “for”, perceba que criamos uma variável chamada “numero” e dizemos que, de acordo com a função “range()” (que permite especificar uma faixa de valores e como será incrementada), a variável iniciará com o valor 1 até o valor que for digitado pelo usuário final. E esta faixa será incrementada de 1 em 1, conforme o último valor especificado dentro da função range(). O conteúdo do “for” será repetido enquanto não atingir o valor máximo que foi digitado pelo usuário final.

Como já podemos perceber, esta forma de laço não é a mais adequada para os momentos de preenchimento de dados de um projeto. Não temos como afirmar quantos usuários serão cadastrados, ou quantos pacientes serão atendidos, este é um caso típico para o “while”. Por outro lado, quando os dados já existem e precisamos simplesmente exibi-los ou até mesmo realizar uma pesquisa dentro deles, o “for” fica como o comando mais utilizado para percorrer todos os elementos de um determinado conjunto.

Usando o “for”, tente montar um código que exiba a tabuada de um valor que será digitado pelo usuário final. Para isso, crie um arquivo chamado “tabuada_for.py”. Não olhe o código a seguir, tente fazer primeiro, ok?

```
tabuada=int(input("Digite um número para exibir a tabuada: "))
print("Tabuada do número ", tabuada)
for valor in range(1,11,1):
    print(str(tabuada) + " x " + str(valor) + " = " + str((tabuada*valor)))
```

Código-fonte 2.19 – Exemplo com o comando “for” para tabuada
Fonte: Elaborado pelo autor (2017)

Repare que, para multiplicarmos os dois valores, utilizamos o operador “*”. No range(), definimos que os valores gerados deverão estar entre 1 e 11, com incremento de 1 em 1. O “11” foi definido porque ele não se inclui no range, ou seja, quando chegar em 11 ele para, portanto, não executará mais o laço quando atingir este valor.

Preparamos muitos códigos, mas muita coisa ainda vem pela frente. Procure refazer, principalmente, os desafios deste capítulo, antes de entrar no próximo, quando, então, explicaremos sobre listas, uma das principais estruturas de armazenamento temporário do Python.

Nos vemos lá!

REFERÊNCIAS

ELMASRI, Ramez; NAVATHE, Shamkant B. **Sistemas de banco de dados**. 6. ed. São Paulo: Pearson Addison Wesley, 2011.

FORBELLONE, Andre Luiz Villar. **Lógica de programação**. 3. ed. São Paulo: Prentice Hall, 2005.

JET BRAINS. **Lightweight IDE for Python & Scientific development**. 2017. Disponível em: <https://www.jetbrains.com/pycharm/download/#section=windows>. Acesso em: 3 nov. 2017.

KUROSE, James F. **Redes de computadores e a Internet**: uma abordagem top-down. 6. ed. São Paulo: Pearson Education do Brasil, 2013.

MAXIMIANO, Antonio Cesar Amaru. **Empreendedorismo**. São Paulo: Pearson Prentice Hall-Brasil, 2012.

PIVA JR., Dilermando **Algoritmos e programação de computadores**. São Paulo: Elsevier, 2012.

PUGA, Sandra. **Lógica de programação e estruturas de dados**. São Paulo: Prentice Hall, 2003.

RHODES, Brandon. **Programação de redes com Python**. São Paulo: Novatec, 2015.

STALLINGS, W. **Arquitetura e organização de computadores**. 8. ed. São Paulo: Prentice Hall Brasil, 2010.