UNIVERSIDADE
FEDERAL DO CEARÁ

Curso de Python

Francisco Vanderlei da Silva Barbosa

Graduando em Matemática Industrial – UFC

vanderleisilvaeng@gmail.com

Matemática Industrial

Setembro - 2017

SUMÁRIO

1. Introdução 1
   1.1. História 1
   1.2. Motivações 1
   1.3. Vantagens 3
2. Sintaxe do Python 5
   2.1. Variáveis e Tipos de Dados 5
   2.1.1. Variáveis 5
   2.1.2. Booleanos 5
   2.1.3. Você Foi Redefinido 6
   2.1.4. Uma Questão de Interpretação 6
   2.1.5. Comentários de Linha Única 6
   2.1.6. Comentários com Múltiplas Linhas 7
   2.1.7. Matemática 7
      2.1.7.1. Potenciação 7
      2.1.7.2. Resto da Divisão Inteira 8
   2.1.8. Revisão 8
3. Atividade: Calculadora de Gorjetas 10
   3.1. Pagando a conta 10
      3.1.1. A Refeição 10
      3.1.2. O Imposto 10
      3.1.3. A Gorjeta 10
      3.1.4. Reatribuição em Uma Única Linha 11
      3.1.5. O Total 11
4. STRINGS E SAÍDAS DO CONSOLE 12
   4.1. Strings 12
      4.1.1. Prática 12
   4.2. Caracteres de escape 12
   4.3. Acesso pelo Índice 13
   4.4. Métodos de string 14
      4.4.1. lower() 14
      4.4.2. upper() 15
      4.4.3. str() 15
   4.5. Notação de Ponto 15
   4.6. Exibindo Strings ..... **Erro! Indicador não definido.**

4.7. Exibindo Variáveis 16
4.8. Concatenação de Strings 16
4.9. Conversão Explícita de Strings 16
4.10. Formatação de Strings com %, Parte 1 17
4.11. Formatação de Strings com %, Parte 2 17
4.12. E Agora, Algo Completamente Familiar 18
5. Atividade: Data e Hora 20
5.1. A Biblioteca datetime 20
5.2. Obtendo a Data e Hora Atuais 20
5.3. Extraindo Informações 20
5.4. Data Quente 21
5.5. Bela Hora 22
5.6. Grand Finale 22
6. CONDICIONAIS E FLUXOS DE CONTROLE 24
6.1. Siga o Fluxo 24
6.2. Compare cuidadosamente! 25
6.3. Compare... Mais de Perto! 25
6.4. Como as Mesas Foram Viradas 26
6.5. Ser e/ou Não Ser 26
6.5.1. And 27
6.5.2. Or 28
6.5.3. Not 28
6.6. Isso e Aquilo (ou Isso, Mas Não Aquilo!) 29
6.7. Misturando e Combinando 29
6.8. Sintaxe das Declarações Condicionais 30
6.9. Se Estiver Tendo... 30
6.10. Problemas com Else, Sinto Por Você, Filho... 31
6.11. Eu Tenho 99 Problemas, mas um Interruptor Não é Um 32
6.12. O Grande Se 33
7. Atividade: PygLatin 35
7.1. Desmembrando 35
7.2. Ahoy! (ou Eu Deveria Dizer Ahoyay!) 35
7.3. Entrada! 35
7.4. Verifique Você Mesmo! 36
7.5. Verifique Você Mesmo... Mais um Pouco 36
7.6. Questionário! 37

7.7. Ay B C 37
7.8. Compreenda 38
7.9. Mova De Volta 38
7.10. Encerrando 39
7.11. Testando, Testando, Essa Coisa Está Ligada? 39
8. FUNÇÕES 41
8.1. O Quão Boas são as Funções? 41
8.2. Junção das Funções 42
8.3. Chamado e Resposta 42
8.4. Parâmetros e Argumentos 42
8.5. Funções Chamando Funções 43
8.6. A Prática Leva à Perfeição 43
8.7. Eu Sei Kung Fu. 44
8.8. Importações Genéricas 44
8.9. Importações de Função 45
8.10. Importações Universais 45
8.11. Aqui Há Dragões 46
8.12. Além das Strings 47
8.12.1. max() 48
8.12.2. min() 48
8.12.3. abs() 48
8.12.4. type() 48
8.13. Revisão: Funções 49
8.14. Revisão: Módulos 49
8.15. Revisão: Funções Embutidas 50
9. Atividade: Tirando Férias 51
9.1. Antes de Começarmos 51
9.2. Planejando Sua Viagem 51
9.3. Chegando Lá 52
9.4. Transporte 52
9.5. Reunindo Tudo 53
9.6. Ei, Nunca Se Sabe! 54
9.7. Planeje Sua Viagem! 54
10. Listas e Dicionários em Python 57
10.1. Introdução às Listas 57
10.2. Acesso por Index 57

10.3. Novos Vizinhos 58
10.4. Comprimento de lista 59
10.5. Fatiamento de Lista 60
10.6. Cortar listas e strings 61
10.7. Mantendo a Ordem 61
10.8. Laço For 62
10.9. Mais com 'for' 63
10.10. Esta próxima parte é chave 64
10.11. Novas Entradas 64
10.12. Alterando sua mente 65
10.13. Retirando algumas coisas 66
10.14. É perigoso ir sozinho! Pegue isso 67
11. Atividade: Um dia no supermercado 69
11.1. Antes de seguirmos à diante 69
11.2. Esta é a chave! 69
11.3. Controle de fluxos e laços 70
11.4. Listas + Funções 71
11.5. Laços em strings 72
11.6. Sua própria Loja! 72
11.7. Investindo em estoque 73
11.8. Mantendo o controle da produção 73
11.9. Algo de valor 74
11.10. Compras no Mercado 75
11.11. Fazendo uma compra 75
11.12. Gerindo o estoque 76
11.13. Vamos verificar! 76
12. O estudante vira professor 77
12.1. Lição número um 77
12.2. Qual é o resultado? 77
12.3. União de dicionários 78
12.4. Rumo ao recorde! 79
12.5. Acima da média é aprovado! 79
12.6. Pondere a média! 80
12.7. Enviando uma carta 81
12.8. Parte do Todo 81
13. Numpy 83

13.1. Conhecendo NumPy 83
13.2. Arrumando Arrays 86
13.3. Matrizes com Listas 87
13.4. Criando Matrizes com NumPy 88
13.4.1. Inserindo elementos no array 88
13.4.2. Deletando elementos do array 90
13.4.3. Dividindo um array 91
14. Referências 92

1. INTRODUÇÃO

1.1. História

O Python foi concebido no final de 1989 e lançado em 1991 por Guido van Rossum no Instituto de Pesquisa Nacional para Matemática e Ciência da Computação (CWI), nos Países Baixos, como um sucessor da "ABC" capaz de tratar exceções e prover interface com o sistema operacional Amoeba através de scripts. Também da CWI, a linguagem ABC era mais produtiva que C, ainda que com o custo do desempenho em tempo de execução. Mas ela não possuía funcionalidades importantes para a interação com o sistema operacional, uma necessidade do grupo. Um dos focos primordiais de Python era aumentar a produtividade do programador. Python é uma linguagem de programação de alto nível, interpretada, de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Atualmente possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation. Apesar de várias partes da linguagem possuírem padrões e especificações formais, a linguagem como um todo não é formalmente especificada.

O nome Python teve a sua origem no grupo humorístico britânico Monty Python, criador do programa Monty Python's Flying Circus, embora muitas pessoas façam associação com o réptil do mesmo nome.

A linguagem foi projetada com a filosofia de enfatizar a importância do esforço do programador sobre o esforço computacional. Prioriza a legibilidade do código sobre a velocidade ou expressividade. Combina uma sintaxe concisa e clara com os recursos poderosos de sua biblioteca padrão e por módulos e frameworks desenvolvidos por terceiros.

1.2. Motivações

Dentre suas várias motivações de uso, temos:

- Qualidade do Software

Por design, o Python implementa uma sintaxe deliberadamente simples e legível, e um modelo de programação altamente coerente. Os recursos da linguagem de maneiras consistentes e limitadas, e resultam naturalmente de um pequeno conjunto de conceitos principais. Isso torna a linguagem mais fácil de aprender, entender e lembrar. Na prática, os programadores não precisam referir-se constantemente a manuais ao ler ou escrever códigos; trata-se de um design ortogonal. Por questões de filosofia, o Python adota uma estratégia um tanto minimalista. Na maneira Python de pensar, explícito é melhor do que implícito e simples é melhor do que complexo. Além desses temas de projeto, o Python inclui ferramentas como módulos e POO, que promovem a reutilização de código naturalmente.

- Produtividade do Desenvolvedor

Durante a grande explosão da Internet, em meados dos anos 90, era difícil encontrar programadores para implementar projetos de software; os desenvolvedores eram solicitados a implementar sistemas com a mesma rapidez com que a Internet evoluia. Agora, na era pós-explosão, de demissões e recessão econômica, o quadro

mudou. Atualmente, as equipes de programação são obrigadas a executar as mesmas tarefas com menos pessoas.

Nesses dois cenários, o Python se distingue como uma ferramenta que permite aos programadores fazer mais com menos esforço. Ela é deliberadamente otimizada para velocidade de desenvolvimento - sua sintaxe simples, tipagem dinâmica, ausência de etapas de compilação e seu conjunto de ferramentas incorporado, permitem que os programadores desenvolvam programas em uma fração de tempo de desenvolvimento necessário para algumas outras ferramentas. O Python aumenta a produtividade do desenvolvedor muitas vezes além do que conseguem as linguagens compiladas ou estaticamente tipadas, como C, C++ e Java. O código em Python normalmente tem de ⅓ a ⅕ do tamanho do código equivalente em C++ ou Java. Isso significa menos digitação, menos depuração e menos manutenção após o desenvolvimento. Nela os programas também são executados imediatamente, sem as cansativas etapas de compilação e vinculação de algumas outras ferramentas. O resultado é que o Python aumenta a produtividade do desenvolvedor muitas vezes além do que se consegue com as linguagens tradicionais.

- Portabilidade do Programa

A maioria dos programas em Python são executados sem necessidade de alteração em todas as principais plataformas. Portar um código Python entre Unix e o Windows, por exemplo, normalmente é apenas uma questão de copiar o código entre as máquinas.

- Bibliotecas de Suporte

O Python vem com um grande conjunto de funcionalidades pré-compiladas e portáveis, conhecidas como biblioteca padrão. Além disso, o Python pode ser estendido com bibliotecas feitas em casa, assim como com um enorme conjunto de software de suporte de aplicativo de outros fornecedores.

- Integração de Componentes

Os scripts em Python podem se comunicar facilmente com outras partes de um aplicativo, usando uma variedade de mecanismos de integração. Essas integrações permitem que o Python seja usado como uma ferramenta de personalização e extensão do produto. Atualmente, o código Python pode chamar bibliotecas C e C++, pode ser chamado a partir de programas em C e C++ e pode ser integrado com componentes Java, dentre outras funcionalidades.

- Facilidade de Uso

Uma linguagem simples, usada para completar tarefas rapidamente. Essa provavelmente é a melhor maneira de pensar no Python como uma linguagem de script. O Python permite que os programas sejam desenvolvidos muito mais rapidamente do que nas linguagens compiladas, como C++. Seu rápido ciclo de desenvolvimento promove um modo de programação exploratório e incremental que precisa ser experimentado para ser apreciado, ele torna as tarefas simples, por sua facilidade de uso e flexibilidade. O Python tem um conjunto de recursos simples, mas permite que os programas se tornem cada vez mais sofisticados, conforme for necessário.

Com o Python pode-se programar serviços para sistemas operacionais; interface gráfica com o usuário (GUI); Scripts de Internet que permite executar uma grande variedade de tarefas em rede; Integração de componentes (pode ser estendido e incorporado em sistemas C e C++); Programação de Banco de Dados e Programação Numérica, com o auxílio da biblioteca Numpy.

1.3. Vantagens

A frente apresento suas vantagens técnicas:

É orientado a objetos: o Python é uma linguagem completamente orientada a objetos. Seu modelo de classes suporta noções avançadas, como polimorfismo, sobrecarga de operadores e herança múltipla; contudo, no contexto da sintaxe e tipagem simples do Python, a POO é notadamente fácil de aplicar.

É portável: a implementação em Python é escrita em ANSI C portável, compila e executa em praticamente todas as principais plataformas em uso atualmente. Como uma lista parcial, o Python está disponível em sistemas Unix, Linux, MS-DOS, MS Windows (95, 98, XP, 7 e 10), Macintosh (OS X), Amiga, AtariST, BeOS, PalmOS, supercomputadores Cray, computadores de grande porte IBM, PDAs executando Linux e muito mais.

É poderoso: da perspectiva de recursos, o Python é um pouco híbrido. Seu conjunto de ferramentas o coloca entre as linguagens de script tradicionais (como Tcl, Scheme e Perl) e as linguagens de desenvolvimento de sistemas (como C, C++ e Java). O Python oferece toda a simplicidade e facilidade de uso de uma linguagem de script, junto com ferramentas de engenharia de software mais avançadas, normalmente encontradas nas linguagens compiladas. Aqui estão alguns dos principais itens que você encontrará nas ferramentas do Python:

- Tipagem Dinâmica

O Python monitora os tipos de objetos que seu programa utiliza ao executar; ele não exige complicadas declarações de tipos e tamanho em seu código. Na verdade, não existe algo como declaração de tipo ou variável em qualquer parte do Python

- Gerenciamento de Memória Automático

O Python aloca e recupera automaticamente os objetos (“coletas de lixo”), quando não são mais usados, e a maioria cresce ou diminui sob demanda. O Python monitora os detalhes da memória de baixo nível, para que você não precise fazer isso.

- Suporte para Programação em Grande Escala

Para a construção de sistemas maiores, o Python inclui ferramentas como módulos, classes e exceções. Essas ferramentas permitem que você organize os sistemas em componentes, utilize POO para reutilizar e personalizar códigos, e tratar de eventos e erros de modo elegante

- Ferramentas Incorporadas

Para processar todos esses tipos de objetos, o Python vem com poderosas operações padrão, incluindo concatenação (junção de coleções), fracionamento (extração de seções), ordenação, mapeamento e muito mais.

- Bibliotecas

Para tarefas mais específicas, o Python também vem com um grande conjunto de bibliotecas previamente desenvolvidas, que suportam tudo, desde correspondência através de expressões regulares até comunicação em rede. As bibliotecas de Python estão no local onde ocorre grande parte das ações em nível de aplicativo.

- Utilitários de Outros Fornecedores

Como o Python é freeware, ele estimula os desenvolvedores a contribuírem com ferramentas previamente desenvolvidas que suportam tarefas além das incorporadas

na linguagem; encontra-se suporte para COM, geração de imagens, ORBs CORBA, XML, fornecedores de bancos de dados e muito mais.

Apesar da variedade de ferramentas no Python, ele mantém uma sintaxe e um design notadamente simples. O resultado é uma ferramenta de programação poderosa que mantém a capacidade de utilização de uma linguagem de script.

Pode ser misturado: os programas em Python podem ser facilmente “colados” em componentes escritos em outras linguagens, de diversas maneiras. Por exemplo, a API C do Python permite que programas em C chamem e sejam chamados por programas em Python, de forma flexível. Isso significa que você pode adicionar funcionalidade no sistema Python, quando necessário, e usar os programas em Python dentro de outros ambientes ou sistemas.

Embora outras linguagens também sejam ferramentas uteis, o Python:

- Tem uma sintaxe mais limpa e um design mais simples do que Perl, o que o torna mais legível e fácil de manter, e ajuda a reduzir erros de programas.
- É mais simples de usar do que Java. O Python é uma linguagem de script, mas a Java herda grande parte da complexidade das linguagens de sistema, como C++.
- É mais simples e fácil de usar do que a C++, e quase não compete com esta; como uma linguagem de script, muitas vezes o Python tem funções diferentes.
- É mais poderoso e mais independente de plataforma do que o Visual Basic. Sua natureza de código-fonte aberto também significa que ele não é controlado por uma única empresa

Especialmente para programas que fazem mais do que varrer arquivos de texto e que talvez tenham de ser lidos por outras pessoas no futuro, acredita-se que o Python é mais adequado do que qualquer outra linguagem de script disponível atualmente. Além disso, a menos que seu aplicativo exija o máximo desempenho, o Python é uma alternativa viável às linguagens de desenvolvimento de sistemas, como C, C++ e Java; o código Python será muito menos difícil de escrever, depurar e manter.

2. SINTAXE DO PYTHON

2.1. Variáveis e Tipos de Dados

O Python é uma linguagem de programação fácil de aprender. Você pode usá-la para criar apps web, jogos, e até mesmo um sistema de buscas!

2.1.1. Variáveis

Criar apps web, jogos e sistemas de busca envolve armazenar e trabalhar com diferentes tipos de dados. Eles fazem isso usando variáveis. Uma variável armazena um dado e dá a ele um nome específico.

Por exemplo:

```
spam = 5
```

A variável `spam` agora armazena o número `5`.

Instruções

1. Atribua o valor `10` à variável `my_variable`.
2. Clique no botão Salvar e Enviar para executar seu código.

2.1.2. Booleanos

Grande! Você acaba de armazenar um número em uma variável. Números são um dos tipos de dados que usamos em programação. Um segundo tipo de dados é chamado booleano.

Um booleano é como um interruptor de luz. Ele pode ter apenas dois valores. Do mesmo modo que um interruptor pode estar apenas ligado ou desligado, um valor booleano pode ser apenas `True` (Verdadeiro) ou `False` (Falso).

Você pode usar variáveis para armazenar booleanos, assim:

```
a = True
b = False
```

Instruções

Atribua os valores seguintes às variáveis correspondentes:

1. `7` à variável `my_int`
2. `1.23` à variável `my_float`
3. `True` à variável `my_bool`

2.1.3. Você Foi Redefinido

Agora você sabe como usar variáveis para armazenar valores.

Diga `my_int = 7`. Você pode mudar o valor de uma variável a "redefinindo", assim:

```
my_int = 3
```

Instruções

Experimente! Mude o valor de `my_int` de `7` para `3` no editor

2.1.4. Uma Questão de Interpretação

O interpretador executa seu código linha por linha, e verifica quaisquer erros.

```
cats = 3
```

No exemplo acima, criamos uma variável `cats` e atribuímos a ela o valor `3`.

Instruções

1. Crie uma variável chamada `spam` e atribua a ela o valor de `True`.
2. Crie uma variável chamada `eggs` e atribua a ela o valor de `False`.

2.1.5. Comentários de Linha Única

O sinal `#` é para comentários. Um comentário é uma linha de texto que o Python não tentará executar como código. É para os humanos lerem.

Comentários tornam seu programa mais fácil de entender. Quando você lê seu código, ou outros querem colaborar com você, eles podem ler seus comentários e entender facilmente o que seu código faz.

Instruções

Escreva um comentário na linha 1. Tenha certeza de que ele começa com #. Ele pode dizer o que você quiser.

2.1.6. Comentários com Múltiplas Linhas

O sinal # marcará um comentário em apenas uma linha. Embora você possa escrever um comentário com mais de uma linha, iniciar cada uma delas com #, pode ser maçante.

Em vez disso, em vez de comentários com múltiplas linhas, você pode incluir o bloco inteiro entre um par de aspas triplas:

```
"""Sipping from your cup 'til it runneth over,
Holy Grail.
"""
```

Instruções

Escreva um comentário com múltiplas linhas no editor. Ele pode ser qualquer texto que você queira!

2.1.7. Matemática

Ótimo! Agora vamos fazer alguma matemática. Você pode somar, subtrair, multiplicar, dividir números assim:

```
addition = 72 + 23
subtraction = 108 - 204
multiplication = 108 * 0.5
division = 108 / 9
```

Instruções

Iguale a variável `count_to` à soma de dois números grandes.

2.1.7.1. Potenciação

Todas essas contas podem ser feitas com uma calculadora, então por que usar Python? Porque que você pode combinar matemática com outros tipos de dados (por exemplo booleanos) e comandos para criar programas úteis. As calculadoras trabalham apenas com números!

Agora vamos trabalhar com expoentes.

```
eight = 2 ** 3
```

No exemplo acima, criamos uma nova variável chamada `eight` e atribuímos a ela o valor `8`, ou o resultado de 2 elevado a 3 (2^3).

Note que usamos `**` em vez de `*` ou operador da multiplicação.

Instruções

Crie uma nova variável chamada `eggs` e use expoentes para igualar `eggs` a 100.

Tente elevar 10 à potência de 2.

2.1.7.2. Resto da Divisão Inteira

Nosso último operador é o resto da divisão inteira. Resto retorna o resto de uma divisão. Então, se você digitar `3 % 2`, ele retornará `1`, porque 2 cabe uma vez em 3, com resto 1.

Instruções

Use resto para igualar `spam` a `1`. Você pode usar quaisquer dois números que gerem um resto `1` para fazer isso.

2.1.8. Revisão

Bom trabalho! Até agora, você aprendeu sobre:

- Variáveis, que armazenam valores para uso posterior
- Tipos de dados, como números e booleanos
- Espaços em branco, que separam declarações
- Comentários, que tornam seu código mais fácil de ler
- Operadores aritméticos, incluindo `+`, `-`, `*`, `/`, `**`, e `%`

Instruções

Vamos colocar nossos conhecimentos para trabalhar.

1. Escreva um comentário de uma única linha na linha 1. Pode ser qualquer coisa! (Tenha certeza de que ele começa com `#`).
2. Iguale a variável `monty` a `True`.

3. Iguale outra variável `python` a `1.234`

3. ATIVIDADE: CALCULADORA DE GORJETAS

3.1. Pagando a conta

3.1.1. A Refeição

Agora vamos aplicar os conceitos da seção anterior a um exemplo do mundo real.

Você acaba de comer em um restaurante, e recebeu esta conta (valores em notação com ponto decimal):

- Custo da refeição: $44.50
- Impostos: 6.75%
- gorjeta: 15%

Você calculará a gorjeta com base no custo total da refeição (incluindo impostos).

Instruções

Primeiro, vamos declarar a variável "refeicao" e atribuir a ela o valor "44.50".

3.1.2. O Imposto

Bom! Agora vamos criar uma variável para o percentual de imposto.

A taxa de imposto em seu recibo é 6.75%. Você terá que dividir 6.75 por 100 para obter a forma decimal da porcentagem.

Instruções

Crie a variável "taxa" e atribua a ela o valor decimal de 6.75%.

3.1.3. A Gorjeta

Bom trabalho! Você foi bem atendido, então vai deixar uma gorjeta de 15% sobre o custo total da refeição, incluindo impostos.

Antes de calcularmos a gorjeta da sua conta, vamos criar uma variável para a conta. Novamente, precisamos usar a forma decimal da gorjeta, então dividimos 15,0 por 100.

Instruções

Atribua a "`gorjeta`" o valor decimal de 15%.

3.1.4. Reatribuição em Uma Única Linha

Ok! Temos as três variáveis de que precisamos para realizar nosso cálculo, e conhecemos alguns operadores aritméticos que podem nos ajudar.

Vimos na Lição 1 que podemos reatribuir os valores das variáveis. Por exemplo, podemos dizer `spam = 7`, e mais tarde mudar de ideia de dizer `spam = 3`.

Instruções

Mude "`refeição`" para o valor de ele mesmo + ele mesmo * impostos. E sim, você pode reatribuir uma variável em termos de ela mesma!

Estamos calculando apenas o custo da refeição e dos impostos aqui. Vamos chegar logo à gorjeta.

3.1.5. O Total

Agora que "`refeição`" contém o custo da refeição mais os impostos, vamos introduzir uma nova variável "`total`" igual a `refeicao + refeicao * gorjeta`.

Instruções

Atribua à variável `total` a soma de `refeicao + refeicao * gorjeta`. Agora você tem o custo total da sua refeição! Depois chame a função "print()" para imprimir a variável "total". No próximo capítulo iremos aprender a trabalhar com "strings".

4. STRINGS E SAÍDAS DO CONSOLE

4.1. Strings

Outro tipo de dados útil é a string. Uma string pode conter letras, números e símbolos.

```
name = "Ryan"
age = "19"
food = "cheese"
```

1. No exemplo acima, criamos uma variavel "name" e atribuimos a ela a string `"Ryan"`.
2. Também precisamos ajustar "age" para `"19"` e "food" para `"cheese"`.

   Obs: As strings devem estar entre aspas.

Instruções

Crie uma nova variável `brian` e atribua a ela a string `"Alô vida!"`.

4.1.1. Prática

Excelente! Vamos praticar um pouco com strings.

Instruções

Atribua às variáveis os valores das frases respectivas:

1. Iguale `caesar` a `"Graham"`
2. Iguale `praline` a `"John"`
3. Iguale `viking` a `"Teresa"`

4.2. Caracteres de escape

Eis alguns caracteres que causam problemas. Por exemplo:

```
'There's a snake in my boot!'
```

Este código falha porque o Python acha que o apóstrofe em `'There's'`encerra a string. Podemos usar a barra invertida para consertar o problema, assim:

```
'There\'s a snake in my boot!'
```

Instruções

Conserte a string no editor!

A string abaixo está errada. Conserte-a usando a barra invertida de escape!

"Neste calor só um copo d'agua bem gelado."

4.3. Acesso pelo Índice

Bom trabalho!

Cada caractere em uma string recebe um número. Este número é chamado de índice. Verifique o diagrama no editor.

```
c = "gatos"[0]
n = "Ryan"[2]
```

1. No exemplo acima, criamos uma nova variável chamada “`c`” e lhe atribuímos o valor `"g"`, o caractere no índice zero da string `"gatos"`.
2. A seguir, criamos uma nova variável chamada “`n`” e lhe atribuímos o valor `"a"`, o caractere no índice dois da string `"Ryan"`.

   No Python, contamos o índice a partir do zero em vez do um.

Instruções

Iguale a variável `quinta_letra` à quinta letra da string "MONTY".

Lembre-se que a quinta letra não está no índice 5. Comece a contar os índices a partir do zero.

```
"""A string "PYTHON" tem seis caracteres,
numerados de 0 a 5, como mostrado abaixo:
+---+---+---+---+---+---+
| P | Y | T | H | O | N |
+---+---+---+---+---+---+
 0   1   2   3   4   5
Entao, se você quisesse o "Y", poderia digitar
"PYTHON"[1] (sempre comece contanto a partir do 0!)"""
quinta_letra = "MONTY"[4]
```

```
print quinta_letra
```

4.4. Métodos de string

Bom trabalho! Agora que sabemos como armazenar strings, vamos ver como podemos modificá-las usando métodos de string.

Métodos de string permitem a você realizar tarefas específicas em strings.

Vamos nos concentrar em quatro métodos de string:

1. `len()`
2. `lower()`
3. `upper()`
4. `str()`

Vamos começar com len(), que obtém o comprimento (número de caracteres) de uma string!

Instruções

1. Crie uma variável chamada `carro` e iguale-a à string `"Azul Marinho"`. Na linha abaixo, digite `len(carro)` depois da palavra `print`, assim: `print(len(carro))`. A saída será o número de letras em `"Azul Marinho"`!

Obs: essa função conta também os espaços em branco da string!

4.4.1. lower()

Bom trabalho!!

Você pode usar o método `lower()`para se livrar de todas as letras maiusculas em suas strings. Voce chama `lower()` assim:

```
"Ryan".lower()
```

que retornara `"ryan"`.

Instruções

Aplique `lower()` em `carro` (depois do `print`).

4.4.2. upper()

Agora sua string está 100% em letras minúsculas! Existe um método similar para deixar uma string totalmente em letras maiúsculas.

Instruções

Aplique `upper()` em carro (depois do `print`) para colocar todos os caracteres da string em maiúsculas!

4.4.3. str()

Agora vamos examinar `str()`, que é um pouco menos direto. O método `str()` converte outros tipos de variáveis em strings! Por exemplo:

```
str(2)
```

converte `2` em `"2"`.

Instruções

1. Crie uma variável `pi` e iguale-a a `3.14`.
2. Chame `str(pi)`, depois de `print`.

4.5. Notação de Ponto

Vamos examinar melhor por que você usa `len(string)` e `str(object)`, mas notação de ponto (como `"String".upper()`) para o resto.

```
lion = "rugido"
len(lion)
lion.upper()
```

Métodos que usam notação de ponto trabalham apenas com strings.

Por outro lado, `len()` e `str()` podem funcionar em outros tipos de dados.

Instruções

1. Chame a função `len()`com o argumento `jogador = “Neymito”`.
2. Após, invoque a função `.upper()` em `jogador`.

4.6. Exibindo Variáveis

Ótimo! Agora que exibimos strings, vamos exibir variáveis

Instruções

1. Declare uma variável chamada `game_of_thrones` e atribua a ela a string `"Starks!"`.
2. Vá em frente e exiba `game_of_thrones.`

4.7. Concatenação de Strings

Você conhece as strings, e conhece operadores aritméticos. Agora vamos combiná-los!

```
print("Viva " + "de " + "Boa!")
```

Isso exibirá a frase `Viva de Boa!`

O operador `+` entre strings as `somará`, uma depois da outra. Note que há espaços dentro das aspas depois de `Viva` e `de` para que possamos fazer a string combinada aparecer como 3 palavras.

Combinar strings dessa forma é chamado concatenação. Vamos agora tentar concatenar algumas strings!

Instruções

Vamos tentar. Exiba as strings concatenadas `"Presunto "`, `"e "`, `"ovos"` na linha 3, como no exemplo acima.

Tenha certeza de incluir os espaços depois de `"Presunto "` e `"e "`.

4.8. Conversão Explícita de Strings

Às vezes você precisa combinar uma string com algo que nao é uma string. Para fazer isso, temos que converter a não-string em uma string.

```
print("Eu tenho " + str(2) + " cocos!")
```

Isso exibirá “`Eu tenho 2 cocos!`”.

O método `str()` converte não-strings em strings. No exemplo acima, você converteu o número `2` em uma string e então concatenou as strings como no exercício anterior.

Agora, tente você mesmo!

Instruções

1. Execute o código como está. Você vai receber um erro!

   print("O valor de pi e de cerca de " + 3.14)

2. Use `str()` para converter `3.14` em uma string. Então, execute o código de novo.

4.9. Formatação de Strings com %, Parte 1

Quando você quiser exibir uma variável com uma string, há um modo melhor do que concatenar strings.

```
name = "Mike"
print("Ola %s" % (name))
```

O operador `%` depois de uma string é usado para combinar uma string com variáveis. O operador `%` substituirá um `%s` na string pela variável string que vem depois dele.

Instruções

Dê uma olhada no código no editor. O que você acha que ele fará? Clique em Salvar e Enviar quando achar que sabe.

torre = “Expresso:”

flutuante = 3.123456

print(“%s %.2f” % (torre, flutuante))

4.10. Formatação de Strings com %, Parte 2

Lembre-se, usamos o operador `%` para substituir os espaçadores temporários `%s` com as variáveis entre parênteses.

```
name = "Mike"
print("Ola %s" % (name))
```

Você precisa do mesmo número de termos "`%s`" em uma string que o número de variáveis entre parênteses:

```
Print("Os %s que %s %s!" % ("Cavaleiros", "Dizem", "Ni"))
# Isso exibia "Os Cavaleiros que Dizem Ni!"
```

Instruções

Agora é sua vez! Temos ___ no código para mostrar o que você precisa mudar!

nome = input("Qual é o seu nome? ")

missao = input("Qual é seu time do coração? ")

cor = input("Qual é sua cor favorita? ")

print("Ah, então seu nome é __, seu time do coração é o __, " \

"e sua cor favorita é __." __ (____, _____, _____))

1. Dentro da string, substitua os três ___ por `%s`.
2. Depois da string, mas antes das três variáveis, substitua o último ___por um `%`.
3. Clique em Salvar e Enviar Código.
4. Responda às perguntas no console conforme elas aparecerem! Digite sua resposta e pressione Enter.

4.11. E Agora, Algo Completamente Familiar

Bom trabalho! Você aprendeu muito nesta unidade, incluindo:

Há três modos de criar strings

```
'Alpha'
"Bravo"
str(3)
```

Métodos de string

```
len("Charlie")
"Delta".upper()
"Echo".lower()
```

Exibição de uma string

```
Print("Foxtrot")
```

Técnicas avançadas de exibição

```
g = "Golf"
h = "Hotel"
print("%s, %s" % (g, h))
```

Instruções

Vamos fechar tudo!

1. Crie a variável `minha_string` e coloque nela qualquer string que quiser.
2. Exiba o comprimento de `minha_string`.
3. Exiba a versão em maiúsculas (`.upper()`) de `minha_string`.

5. ATIVIDADE: DATA E HORA

5.1. A Biblioteca datetime

Muitas vezes, você quer registrar o momento em que algo aconteceu. Podemos fazer isso no Python usando "datetime".

Aqui, usaremos "datetime" para exibir a data e hora em um formato claro.

Instruções

Clique em Salvar e Enviar Código para continuar.

from datetime import datetime

5.2. Obtendo a Data e Hora Atuais

Podemos usar uma função chamada "datetime.now()" para recuperar a data e hora atuais.

```
from datetime import datetime

print(datetime.now())
```

A primeira linha importa a biblioteca "datetime" para que possamos usá-la.

A segunda linha exibe a data e hora atuais.

Instruções

1. Crie uma variável chamada "agora" e armazene nela o resultado de "datetime.now()".
2. então, exiba na tela o valor de "agora".

5.3. Extraindo Informações

Note como a saída se parece com `2013-11-25 23:45:14.317454`. E se você não quiser a data e hora completos?

```
from datetime import datetime
agora = datetime.now()
```

```
ano_atual = agora.year
mes_atual = agora.month
dia_atual = agora.day
```

Você já tem as primeiras duas linhas.

Na terceira linha, tomamos o ano (e apenas o ano) da variável `agora` e o armazenamos em `ano_atual`.

Na quarta e quinta linhas, armazenamos o mês e dia de `agora`.

Instruções

1. Em uma nova linha, digite `print(agora.year)`. Tenha certeza de fazer isso depois de gerar a variável `agora`!
2. Então, exiba `agora.month`.
3. Finalmente, exiba `agora.day`.

5.4. Data Quente

E se quisermos exibir a data de hoje no seguinte formato? `mm/dd/yyyy`. Vamos usar substituição de strings de novo!

```
from datetime import datetime
now = datetime.now()

print('%s-%s-%s' % (now.year, now.month, now.day))
# irá imprimir: 2014-02-19
```

Lembre-se que o operador `%` preencherá os espaços temporários `%s` na string à esquerda com as strings nos parênteses à direita.

No exemplo acima, exibimos `2014-02-19` (supondo que hoje seja 19 de fevereiro de, 2014), mas você vai exibir `19/02/2014`.

Instruções

Exiba a data atual na forma `dd/mm/yyyy`.

1. Mude a string de modo que ela use um caractere "`/`" entre os espaços temporários `%s` em vez de um caractere "`-`".
2. Rearranje os parâmetros à direita do operador `%` de modo que você exiba `now.day`, `now.month`, e `now.year`.

5.5. Bela Hora

Bom trabalho! Vamos fazer o mesmo para a hora, minuto e segundo.

```
from datetime import datetime
agora = datetime.now()

print(agora.hour)
print(agora.minute)
print(agora.second)
```

No exemplo acima, simplesmente exibimos a hora atual, então o minuto atual, então o segundo atual.

Podemos usar novamente a variável `agora` para exibir o tempo.

Instruções

Similar ao último exercício, exiba a hora atual no formato `hh:mm:ss`.

1. Mude a string que você está exibindo de modo que tenha um caractere "`:`" entre os espaçadores temporários `%s`.
2. Mude as três strings que você está exibindo de mês, dia e ano para `agora.hour`, `agora.minute`, e `agora.second`.

5.6. Grand Finale

Conseguimos exibir a data e hora separadamente de modo muito estético. Vamos combinar os dois!

```
from datetime import datetime
agora = datetime.now()

print('%s/%s/%s' % (agora.day, agora.month, agora.year))
print('%s:%s:%s' % (agora.hour, agora.minute, agora.second))
```

O exemplo acima exibirá a data, e então, em uma linha separada, a hora.

Vamos exibi-los todos na mesma linha com uma única declaração `print`!

Instruções

Exiba a data e hora juntos, no formato: `dd/mm/yyyy hh:mm:ss`.

Para começar, mude a string a esquerda do operador `%`.

1. Garanta que ela tem 6 `%s` espaçadores temporários.

2. Coloque barras e dois pontos e um espaço entre os espaçadores temporários para que eles se encaixem no formado acima.

3. Então, mude as variáveis entre parênteses para a direita do operador `%`.

4. Posicione as variáveis de modo que `agora.day, agora.month, agora.year` fiquem à frente de `agora.hour, agora.minute, agora.second`.

6. CONDICIONAIS E FLUXOS DE CONTROLE

6.1. Siga o Fluxo

Como na vida real, às vezes queremos que nosso código seja capaz de tomar decisões.

Os programas Python que escrevemos até agora têm ideias fixas: eles podem somar dois números ou `exibir` alguma coisa, mas não têm a capacidade de escolher um desses resultados em lugar do outro.

O controle de fluxo nos dá a capacidade de escolher entre resultados com base no que está acontecendo no programa.

Instruções

Examine o código no editor. Você verá o tipo de programa que poderá escrever quando tiver dominado o controle de fluxo. Clique em Salvar e Enviar para ver o que acontece!

```
def clinica():
    print("Voce acabou de entrar na clinica!")
    print("Voce entra pela porta a esquerda (e) ou a direita (d)?")
    answer = input("Digite E (esquerda) ou D (direita) e pressione 'Enter'.").lower()
    if answer == "e" or answer == "left":
        print("Esta é a sala de Abuso Verbal, seu monte de caca de papagaio!")
    elif answer == "d" or answer == "right":
        print("É claro que esta é a Sala das Discussões. Eu já disse isso!"
    else:
        print("Voce não escolheu esquerda ou direita. Tente de novo."
        clinica()
```

6.2. Compare cuidadosamente!

Vamos começar com o aspecto mais simples do controle de fluxo: comparadores. Existem seis:

1. Igual a ("==")
2. Diferente de ("!=")
3. Menor do que ("<")
4. Menor ou igual a ("<=")
5. Maior que (">")
6. Maior ou igual a (">=")

Comparadores verificam se um valor é (ou não) igual, maior do que (ou igual a), ou menor que (ou igual a) outro valor.

Note que "==" compara se duas coisas são iguais, e "=" atribui um valor a uma variável.

Instruções

Iguale cada variável a `True` ou `False`, dependendo de qual você acha que será o resultado. Por exemplo, `1 < 2` será `True`, porque um é menor do que dois.

1. Iguale `bool_one` ao resultado de `17 < 328`
2. Iguale `bool_two` ao resultado de `100 == (2 * 50)`
3. Iguale `bool_three` ao resultado de `19 <= 19`
1. Iguale `bool_four` ao resultado de `-22 >= -18`
2. Iguale `bool_five` ao resultado de `99 != (98 + 1)`

6.3. Compare... Mais de Perto!

Excelente! Parece que você está confortável com expressões e comparadores básicos.

Mas e quanto a expressões e comparadores extremos?

Instruções

Vamos estudar os comparadores novamente com expressões mais complexas. Atribua um valor `True` ou `False` a cada variável, dependendo de qual você acha que será o resultado.

1. Iguale `bool_one` ao resultado de `(20 - 10) > 15`
2. Iguale `bool_two` ao resultado de `(10 + 17) == 3**16`
3. Iguale `bool_three` ao resultado de `1**2 <= -1`
4. Iguale `bool_four` ao resultado de `40 * 4 >= -4`
5. Iguale `bool_five` ao resultado de `100 != 10**2`

6.4. Como as Mesas Foram Viradas

Comparações resultam em True (Verdadeiro) ou False (Falso), que são valores booleanos.

```
# me faça verdadeiro!
bool_one = 3 < 5
```

Vamos dificultar: vamos dar a você o booleano, e você escreverá a expressão, como no exemplo acima.

Instruções

Escreva uma expressão que avalie cada valor booleano no editor.

Lembre-se, os comparadores são: `==`, `!=`, `>`, `>=`, `<`, e `<=`.

Use pelo menos três comparadores diferentes!

Não use simplesmente `True` e `False`! Isso é trapaça!

6.5. Ser e/ou Não Ser

Operadores booleanos comparam declarações e resultam em valores booleanos. Há três operadores booleanos:

1. `and` (e), que verifica se as duas afirmações são `True`;
2. `or` (ou), que verifica se pelo menos uma das afirmações é `True`;
3. `not` (não), que gera o oposto da afirmação.

   Vamos estudar os operadores um por um.

Instruções

Examine a tabela verdade no editor. Não se preocupe se não a entender completamente ainda — você vai entendê-la no fim desta seção!

"""

Operadores Booleanos

True and True e True

True and False e False

False and True e False

False and False e False

True or True e True

True or False e True

False or True e True

False or False e False

Not True e False

Not False e True

"""

Clique em Salvar e Enviar para continuar.

6.5.1. And

O operador booleano `and` (e) retorna `True` (verdadeiro) quando as expressões dos dois lados do `and` são verdadeiros. Por exemplo:

- `1 < 2 and 2 < 3` é `True`;
- `1 < 2 and 2 > 3` é `False`.

Instruções

Vamos praticar com `and`. Atribua a cada variável o valor booleano apropriado.

1. Iguale `bool_one` ao resultado de `False and False`
2. Iguale `bool_two` ao resultado de `-(-(-(-2))) == -2 and 4 >= 16**0.5`
3. Iguale `bool_three` ao resultado de `19 % 4 != 300 / 10 / 10 and False`
4. Iguale `bool_four` ao resultado de `-(1**2) < 2**0 and 10 % 10 <= 20 - 10 *2`
5. Iguale `bool_five` ao resultado de `True and True`

6.5.2. Or

O operador booleano `or` (ou) retorna `True` (verdadeiro) quando pelo uma das expressões ligadas por `or` for verdadeira. Por exemplo:

- `1 < 2 or 2 > 3` é `True`;
- `1 > 2 or 2 > 3` é `False`.

Instruções

Hora de praticar com or!

1. Iguale `bool_one` ao resultado de `2**3 == 108 % 100 ou 'Cleese' == 'King Arthur'`
2. Iguale `bool_two` ao resultado de `True ou False`
3. Iguale `bool_three` ao resultado de `100**0.5 >= 50 ou False`
4. Iguale `bool_four` ao resultado de`True ou True`
5. Iguale `bool_five` ao resultado de`1**100 == 100**1 ou 3 * 2 * 1 != 3 + 2+ 1`

6.5.3. Not

O operador booleano `not` retorna `True` para declarações falsas e `False` para verdadeiras.

Por exemplo:

- `not False` resultará em `True`, enquanto `not 41 > 40` retornará `False`.

Instruções

Vamos praticar um pouco com `not`.

1. Iguale `bool_one` ao resultado de `not True`
2. Iguale `bool_two` ao resultado de `not 3**4 < 4**3`
3. Iguale `bool_three` ao resultado de `not 10 % 3 <= 10 % 2`
4. Iguale `bool_four` ao resultado de `not 3**2 + 4**2 != 5**2`
5. Iguale `bool_five` ao resultado de `not not False`

6.6. Isso e Aquilo (ou Isso, Mas Não Aquilo!)

Operadores booleanos não são simplesmente avaliados da esquerda para a direita. Assim como os operadores aritméticos, há uma ordem na execução dos operadores booleanos:

1. `not` é o primeiro a ser avaliado;
2. `and` é o segundo a ser avaliado;
3. `or` é o último a ser avaliado.

Por exemplo, `True or not False and False` retorna `True`.

Parênteses `()` garantem que suas expressões sejam avaliadas na ordem que você quer. Qualquer coisa entre parênteses é avaliada como uma unidade separada.

Instruções

Atribua `True` ou `False`, conforme apropriado, a `bool_one` até `bool_five`.

1. Iguale `bool_one` ao resultado de `False or not True and True`
2. Iguale `bool_two` ao resultado de `False and not True or True`
3. Iguale `bool_three` ao resultado de `True and not (False or False)`
4. Iguale `bool_four` ao resultado de `not not True or False and not True`
5. Iguale `bool_five` ao resultado de `False or not (True and True)`

6.7. Misturando e Combinando

Bom trabalho! Estamos quase acabando com os operadores booleanos.

```
# me faça falso!
bool_one = (2 <= 2) and "Alpha" == "Bravo"
```

Instruções

Desta vez daremos o resultado esperado, e você usará alguma combinação de operadores booleanos para atingir esse resultado.

Lembre-se os operadores booleanos são `and`, `or`, e `not`. Use cada um pelo menos uma vez!

6.8. Sintaxe das Declarações Condicionais

`if` é uma declaraçao condicional que executa um código especificado depois de verificar se sua expressão é `True` (verdadeira).

Eis um exemplo da sintaxe da declaração `if`:

```
if 8 < 9:
    print("Eight is less than nine!")
```

Neste exemplo, `8 < 9` e é a expressão verificada e `print "Eight is less than nine!"` ("Oito é menor do que nove") é o código especificado.

Instruções

Se você acha que a declaração 'print' será exibida no console, atribua o valor `'Y'` a `response` caso contrário, atribua o valor `'N'` a `response`.

response = 'Y'

answer = "Left"

if answer == "Left":

    print("Esta e a Sala de Abuso Verbal, seu monte de caca de papagaio!")

A declaracao print acima sera exibida no console?

Iguale response a 'Y' se achar que sim, ou 'N' se achar que nao.

6.9. Se Estiver Tendo...

Vamos praticar um pouco o uso das declarações `if`. Lembre-se, a sintaxe é assim:

```
if some_function():
```

```
    # linha um do bloco
    # linha dois do bloco
    # etc.
```

No exemplo acima, caso `some_function()` retorne `True`, então o bloco recuado de código depois dela será executado. Caso ela retorne `False`, o bloco recuado será pulado.

Além disso, note os dois pontos no final da declaração `if`. Nós os adicionamos para você, mas eles são importantes.

Instruções

No editor você verá duas funções. Não se preocupe se algo não for familiar. Vamos explicar tudo em breve.

1. Substitua o underline na primeira função por uma expressão que retorne `True`.
2. Substitua o underline na segunda função por uma expressão que retorne `True`.

Se você fizer isso corretamente, `"Successo #1"` e `"Successo #2"` serão exibidos.

```
def using_control_once():
    if _______:
        return "Successo #1"

def using_control_again():
    if _______:
        return "Successo #2"

print using_control_once()
print using_control_again()
```

6.10. Problemas com Else, Sinto Por Você, Filho...

A declaração `else` complementa a declaração `if`. Um par `if`/`else` diz: "Se esta expressão for verdadeira, execute o bloco de código recuado; caso contrário, execute o código depois da declaração else."

Ao contrário de `if`, `else` não depende de uma expressão. Por exemplo:

```
if 8 > 9:
    print "I don't printed!"
else:
    print "I get printed!"
```

Instruções

Complete as declarações `else` à direita. Note o recuo de cada linha!

```
answer = "E so um arranhao!"


def black_knight():
    if answer == "E so um arranhao!":
        return True
    else:
        return _____ # Tenha certeza de que isso retorna False


def french_soldier():
    if answer == "Va embora, ou vou zombar de você mais uma vez!":
        return True
    else:
        return _____  # Tenha certeza de que isso retorna False
```

6.11. Eu Tenho 99 Problemas, mas um Interruptor Não é Um

"Elif" é uma abreviação para "else if." Isso significa exatamente o que parece: "caso contrário, se a expressão a seguir for verdadeira, faça isto!"

```
if 8 > 9:
    print("I don't get printed!")
elif 8 < 9:
    print("I get printed!")
else:
    print("I also don't get printed!")
```

No exemplo acima, a declaração `elif` é verificada apenas se a declaração `if` original for `False`.

Instruções

1. Na linha 3, preencha a declaração `if` para verificar se (if) `answer` é maior que 5.
2. Na linha 5, preencha `elif` de modo que a função resulte em `-1` se `answer` for menor que 5.

```
answer = 4

def greater_less_equal_5(answer):
    if _________:
        return 1
    elif _________:
        return -1
    else:
        return 0


print greater_less_equal_5(4)
print greater_less_equal_5(5)
print greater_less_equal_5(6)
```

6.12. O Grande Se

Excelente trabalho! Eis o que você aprendeu nesta unidade:

Comparadores

```
3 < 4
5 >= 5
10 == 10
12 != 13
```

Operadores booleanos

```
True or False
(3 < 4) and (5 >= 5)
this() and not that()
```

Declaracoes condicionais

```
if this_might_be_true():
    print "This really is true."
elif that_might_be_true():
    print "That is true."
else:
    print "None of the above."
```

Vamos para o grand finale.

Instruções

Escreva uma declaração `if` em `the_flying_circus()`. Ela deve incluir:

1. Declarações `if`, `elif`, e `else`;
2. Pelo menos um `and`, `or`, ou `not`;
3. Um comparador (`==`, `!=`, `<`, `<=`, `>`, or `>=`);
4. Finalmente, `the_flying_circus()`deve `retornar True` quando avaliado.

Não se esqueça de incluir um `:`depois das suas declarações `if`!

```
# Tenha certeza que the_flying_circus() retorna True

def the_flying_circus():
    if _____:    # Comece seu codigo aqui!
        return 'arroz integral'    # Nao esqueca de recuar
        # o codigo dentro deste bloco!
    Elif _____:
        # Continue aqui.
        # Voce vai querer adicionar também a declaração else!
    else:
        return 0
```

7. ATIVIDADE: PYGLATIN

7.1. Desmembrando

Agora, vamos usar o que aprendemos até agora e escrever um tradutor de Pig Latin.

Pig Latin é um jogo linguístico, em que você move a primeira letra da palavra para o final e adiciona "ay". Então, "Python" se torna "ythonpay". Eis as etapas que você deverá realizar para escrever um tradutor de Pig Latin em Python.

1. Peça ao usuário para inserir uma palavra.
2. Garanta que o usuário inseriu uma palavra válida.
3. Converta a palavra para o Pig Latin.
4. Exiba o resultado da tradução.

7.2. Ahoy! (ou Eu Deveria Dizer hoyAay!)

Vamos nos aquecer exibindo uma mensagem de boas-vindas para os usuários do nosso tradutor.

Instruções

1. Exiba `print` a frase "Pig Latin".

7.3. Entrada!

A seguir precisamos de respostas do usuário.

```
name = input("Qual é seu nome? ")
print(name)
```

No exemplo acima, `input()` aceita uma string, a exibe, e então espera que o usuário digita mais alguma coisa e pressione Enter (ou Return).

No interpretador, Python perguntará:

```
Qual é seu nome?
```

Quando você digitar seu nome e pressionar Enter, ele será armazenado em `name`.

Instruções

1. Use `input("Insira uma palavra: ")` para pedir ao usuário para digitar uma palavra. Salve os resultados de `input()` em uma variável chamada `original`. Execute o código.
2. Digite uma palavra na janela do console e pressione Enter (ou Return).

```
print('Bem-vindo ao Tradutor de Pig Latin!')

# seu código começa aqui!
```

7.4. Verifique Você Mesmo!

A seguir, precisamos garantir que o usuário realmente digitou alguma coisa.

```
vazio = ""
if len(vazio) > 0:
    # Execute este bloco.
    # Talvez exibir alguma coisa?
else:
    # A string deve estar em branco.
```

Podemos verificar se a string do usuário realmente contém caracteres!

Instruções

Escreva uma declaração `if` que verifique que a string tem caracteres.

1. Adicione uma declaração `if` que verifique se `len(original)` é maior do que zero. Não se esqueça do "`:`" no final da declaração `if`!
2. Se a string realmente contiver alguns caracteres, `exiba` a palavra do usuário.
3. Caso contrário, (ou seja, uma declaração `else:`), exiba "Vazio".

Você vai querer executar seu código várias vezes, testando strings vazias e strings com caracteres. Quando estiver confiante que seu código funciona, prossiga para o próximo exercício.

7.5. Verifique Você Mesmo... Mais um Pouco

Agora sabemos que temos uma string com conteúdo. Vamos ser ainda mais detalhistas.

```
x = "J123"
x.isalpha()  # False
```

Na primeira linha, criamos uma string com letras e números.

A segunda linha então executa a função `isalpha()` que retorna `False`, já que a string contém caracteres que não são letras.

Vamos garantir que a palavra que o usuário inseriu contém apenas letras. Podemos usar `isalpha()` para verificar isso! Por exemplo:

Instruções

Use `and` para adicionar uma segunda condição à sua declaração `if`. Além da sua verificação anterior de que a string contém caracteres, você deve também usar `.isalpha()` para garantir que ela contém apenas letras.

Não se esqueça de manter os dois pontos no fim da declaração `if`!

7.6. Questionário!

Quando terminar uma parte do seu programa, é importante testá-lo diversas vezes, usando uma variedade de entradas.

Instruções

Pare por um momento para testar seu código atual. Teste algumas entradas que devem passar e algumas que devem falhar. Insira algumas strings que contém caracteres não alfabéticos e uma string em branco.

Quando estiver convencido que seu código está pronto, prossiga no curso.

7.7. Ay B C

Agora podemos nos preparar para começar a traduzir para Pig Latin! Vamos revisar as regras da tradução:

Você move a primeira letra da palavra para o fim, e então adiciona o sufixo 'ay'. Exemplo: python -> ythonpay

Vamos criar uma variável para armazenar nosso sufixo de tradução.

Instruções

Crie uma variável chamada `pyg` e a iguale ao sufixo `'ay'`.

7.8. Compreenda

Vamos simplificar as coisas colocando as letras em nossa palavra em minúsculas.

```
the_string = "Hello"
the_string = the_string.lower()
```

A função `.lower()` não modifica a própria string, ela simplesmente retorna uma versão em letras minúsculas. No exemplo acima, armazenamos o resultado na mesma variável.

Também precisamos capturar a primeira letra da palavra.

```
Primeira_letra  = the_string[0]
Segunda_letra = the_string[1]
Terceira_letra  = the_string[2]
```

Lembre-se que começamos a contar a partir do zero, e não do um, então acessamos a primeira letra solicitando `[0]`.

Instruções

Dentro da sua declaração `if`:

1. Crie uma nova variável chamada `palavra` que contém a conversão de `original` para letras minúsculas (`.lower()`-case).
2. Crie uma nova variável chamada `primeira` que contém `palavra[0]`, a primeira letra de `palavra`.

7.9. Mova De Volta

Agora que a primeira letra foi armazenada, precisamos adicionar tanto a letra quanto a string armazenados em `pyg` ao final da string original.

Lembra-se de como concatenar (ou seja, somar) strings?

```
greeting = "Hello "
name = "D. Y."
welcome = greeting + name
```

Instruções

Em uma nova linha depois que você tiver criado a variável `primeira`:

Crie uma nova variável chamada `nova_palavra` e a iguale à concatenação de `palavra`, `primeira`, e `pyg`.

7.10. Encerrando

Bom trabalho! Entretanto, agora temos a primeira letra parecendo tanto no começo quanto perto do fim.

```
s = "Charlie"

print s[0]
# irá imprimir "C"

print s[1:4]
# irá imprimir "har"
```

1. Primeiro, criamos uma variável `s` e damos a ela a string `"Charlie"`
2. A seguir, acessamos a primeira letra de `"Charlie"` usando `s[0]`. Lembre-se que as posições das letras começam em 0.
3. Então, acessamos uma fatia de `"Charlie"` usando `s[1:4]`. Isto retorna tudo a partir da letra na posição 1 até a posição 3.

   Vamos dividir a string do mesmo modo que fizemos no terceiro exemplo acima.

Instruções

Iguale `nova_palavra` à fatia do primeiro índice até o fim. Use `[1:len(nova_palavra)]` para isso.

7.11. Testando, Testando, Essa Coisa Está Ligada?

Isso! você deve ter um tradutor de Pig Latin totalmente funcional. Teste seu código detalhadamente para garantir que tudo esteja funcionando corretamente.

Você também vai querer eliminar quaisquer declarações `print`que estivesse usando para ajudar a depurar as etapas intermediárias do seu código. Agora também é uma boa hora para adicionar alguns comentários! Garantir que seu código está limpo, comentado e totalmente funcional é tão importante quanto escrevê-lo em primeiro lugar.

Instruções

Quando tiver certeza que seu tradutor está funcionando exatamente como você quer execute o mesmo.

pyg = 'ay' #sufixo do tradutor pyg

```
original = "Hello" #palavra a ser traduzida
word = original.lower() #transforma as letras da palavra em minusculas
first = word[0] #separar o primeiro caractere da palavra
new_word = word+first+pyg #concatenar os passos citados a cima
new_word = new_word[1:len(new_word)] #retirar o primeiro caractere da palavra

if len(new_word) > 0 and new_word.isalpha(): #testando se a palavra e ums string
    print new_word #imprimir na tela a palavra traduzida para pyg
else:
    print 'empty' #se nao for uma string retorna 'empty'
```

8. FUNÇÕES

8.1. O Quão Boas são as Funções?

Você dever ter considerado a situação em que reutiliza e um trecho de código, apenas com valores diferentes. Em vez de reescrever todo o código, é muito mais limpo definir uma função, que pode então ser usada repetidamente.

Instruções

Verifique o código no editor. Se você tiver terminado o projeto [Calculadora de Gorjeta][1], se lembrará de calcular impostos e gorjeta em um trecho do programa. Aqui, você pode ver que definimos duas funções: `tax` para calcular o imposto sobre uma conta, e `tip` para calcular a gorjeta.

Veja quanto do código você entende à primeira vista (logo o explicaremos todo). Quando estiver pronto, clique em Salvar e Enviar para continuar.

```
def tax(bill):
    """Soma 8% de imposto a uma conta de restaurante."""
    bill *= 1.08
    print("Com imposto: %.2f" % bill)
    return bill


def tip(bill):
    """Soma uma gorjeta de 15% a uma conta de restaurante."""
    bill *= 1.15
    print("com gorjeta de: %.2f" % bill)
    return bill


meal_cost = 100
meal_with_tax = tax(meal_cost)
meal_with_tip = tip(meal_with_tax)
```

8.2. Junção das Funções

Funções são definidas com três componentes:

1. O header (cabeçalho), que inclui a palavra-chave "def", o nome da função e quaisquer parâmetros de que a função precise. Por exemplo:

```
def hello_world(): // There are no parameters
```

2. Um comentário opcional que explica o que a função faz.

```
"""Prints 'Hello World!' to the console."""
```

3. O corpo, que descreve os procedimentos que a função executa. O corpo é feito em parágrafos, assim como as declarações condicionais.

```
Print("Hello World!")
```

Eis aqui a função completa:

```
def hello_world():
    """Prints 'Hello World!' to the console."""
    print "Hello World!"
```

Instruções

Vá em frente e crie uma função, `spam`, que exibe (`print`) a string `"Ovos!"` no console. Não se esqueça de incluir um comentário à sua escolha (entre aspas triplas!).

8.3. Chamado e Resposta

Depois que uma função é definida, ela deve ser chamada para ser implementada. No exercício anterior, `spam()` na última linha diz ao programa para procurar pela função chamada `spam` e executar o código dentro dela.

Instruções

Vamos configurar uma função, `square`. Aplique-a no número `10`(colocando `10` entre os parênteses de `square()`) na linha 9!

8.4. Parâmetros e Argumentos

Vamos reexaminar a primeira linha que definiu `square` no exercício anterior:

```
def square(n):
```

`n` é um parâmetro de `square`. Um parâmetro age como um nome de variável para um argumento inserido. No exemplo anterior, chamamos `square` com o argumento `10`. Neste caso, `n` tem valor `10`.

Uma função pode exigir quantos parâmetros você quiser, mas quando você chama a função, geralmente deve inserir um número correspondente de argumentos.

Instruções

Verifique a função no editor, `power`. Ela deve ter dois argumentos, uma base e um expoente, e elevar o primeiro à potência do segundo. No momento, ela não funciona, porque os parâmetros estão faltando.

Substitua os ___s com os parâmetros `base` e `exponent` e chame `power` com uma `base` de `37` e uma potência (`power`) de `4`.

8.5. Funções Chamando Funções

Vimos funções que podem exibir texto ou realizar aritimética simples, mas as funções podem ser muito mais poderosas do que isso. Por exemplo, uma função pode chamar outra função:

```
def fun_one(n):
    return n * 5

def fun_two(m):
    return fun_one(m) + 7
```

Instruções

Vamos examinar as duas funções no editor: `one_good_turn` (que soma `1` ao número que recebe como argumento) e `deserves_another` (que soma `2`).

Mude o corpo de `deserves_another` de modo que ele sempre some `2` à saída de `one_good_turn`.

8.6. A Prática Leva à Perfeição

Vamos criar mais algumas funções só para garantir.

```
def shout(phrase):
    if phrase == phrase.upper():
        return "YOU'RE SHOUTING!"
    else:
```

```
        return "Can you speak up?"

shout("I'M INTERESTED IN SHOUTING")
```

O exemplo acima está aqui apenas para ajudá-lo a lembrar como as funções são estruturadas.

Não se esqueça dos dois pontos no final da definição da função!

Instruções

1. Primeiro, defina uma função chamada `cubo` que toma um argumento chamado `number`. Não se esqueça dos parênteses e dos dois pontos!
2. Faça essa função retornar o cubo daquele número (ou seja, aquele número multiplicado por si mesmo, e depois multiplicado mais uma vez por si mesmo).
3. Defina uma segunda função chamada `by_three` que toma um argumento chamado `number`.
4. Se (`if`) esse número for divisível por 3, `by_three` deve chamar `cube(number)` e retornar seu resultado. Caso contrário, `by_three` deve retornar falso (`return False`).

Não se esqueça que as declarações `if` e `else` precisam de um “`:`” ao final daquela linha!

8.7. Eu Sei Kung Fu.

Lembra-se de `import this` do primeiro exercício do curso? Aquilo foi um exemplo de importação de um módulo. Um módulo é um arquivo que contém definições — incluindo variáveis e funções — que você pode usar depois de importadas.

Instruções

Antes de tentarmos fazer qualquer importação, vamos ver o que o Python já sabe sobre raízes quadradas. Peça ao Python para

```
Print(sqrt(25))
```

que esperamos ser igual a cinco.

8.8. Importações Genéricas

Você viu? O Python disse: "NameError: name 'sqrt' is not defined." O Python não sabe o que são raízes quadradas — ainda.

Há um módulo do Python chamado `math` que inclui diversas variáveis e funções úteis, e `sqrt()` é uma dessas funções. Para acessar `math`, tudo o que você precisar é a palavra-chave `import`. Quando você simplesmente importa um módulo desse modo, isso é chamado importação genérica.

Instruções

Você precisa fazer duas coisas aqui:

1. Digitar `import math`.
2. Inserir `math.` antes de `sqrt()` para que tenha a forma `math.sqrt()`. Isso diz ao Python não apenas para `importar math`, mas para obter a função `sqrt()` de dentro de `math`.

Então, calcule novamente.

8.9. Importações de Função

Bom trabalho! Agora, o Python sabe como tirar a raiz quadrada de um número.

Entretanto, nós só precisamos da função `sqrt`, e pode ser frustrante ficar digitando `math.sqrt()` o tempo todo.

É possível importar apenas certas variáveis ou funções de um dado módulo. Obter apenas uma única função de um módulo é chamado importação de função, e é feito com a palavra-chave `from`:

```
from module import function
```

Agora, você pode simplesmente digitar `sqrt()` para obter a raiz quadrada de um número — nada mais de `math.sqrt()`!

Instruções

Dessa vez, vamos importar apenas a função `sqrt` de `math`. (Você não precisa do `()` depois de `sqrt` em `from math import sqrt`).

8.10. Importações Universais

Ótimo! Encontramos um modo de selecionar as variáveis e funções que queremos dos módulos.

E se ainda quisermos todas as funções e variáveis de um módulo mas não quisermos digitar constantemente `math.`?

A importação universal pode tratar disso para você. A sintaxe é:

```
from module import *
```

Instruções

Use o poder de `from module import *` para importar tudo do módulo `math.`

8.11. Aqui Há Dragões

Importações universais podem parecer ótimas na superfície, mas não são uma boa ideia por um motivo muito importante: elas enchem seu programa com uma tonelada de nomes de funções e variáveis sem a segurança desses nomes ainda estarem associados com o(s) módulo(s) de onde eles vieram.

Se você tiver uma função própria chamada `sqrt` e `importar math`, sua função está segura: há o seu `sqrt` e há `math.sqrt`. Entretanto, se você usar `from math import *`, você terá um problema: especificamente, duas funções diferentes com exatamente o mesmo nome.

Mesmo se suas próprias definições não entrarem em conflito direto com os nomes de módulos importados, se você usar `import *` de diversos módulos ao mesmo tempo, não será capaz de descobrir que variável ou função veio de onde.

Por essas razões, é melhor usar ou `import module` e digitar `module.name` ou simplesmente usar `import` para variáveis e funções específicas de diversos módulos conforme necessário.

Instruções

O código no editor mostrará a você tudo o que está disponível no módulo `math`.

Clique em Salvar e Enviar Código para ver (você verá `sqrt`, juntamente com outras coisas úteis, como `pi`, `factorial`, e funções trigonométricas).

```
import math            # Importa o modulo math

everything = dir(math) # Converte tudo em uma lista de coisas de math

print(everything)      # Exibe tudo!
```

8.12. Além das Strings

Agora que você entende o que são funções e como importar módulos, vamos examinar algumas das funções embutidas no Python (não são necessários módulos).

Você já conhece algumas das funções embutidas que usamos com strings, como `.upper()`, `.lower()`, `str()`, e `len()`. Elas são ótimas para trabalhar com strings, mas e quanto a algo um pouco mais analítico?

Instruções

O que você acha que o código no editor fará? Clique em Salvar e Enviar Código quando achar que tem uma ideia.

```
def biggest_number(args):
    print max(args)
    return max(args)


def smallest_number(args):
    print min(args)
    return min(args)


def distance_from_zero(args):
    print abs(arg)
    return abs(arg)




biggest_number(-10, -5, 5, 10)
smallest_number(-10, -5, 5, 10)
distance_from_zero(-10)
```

8.12.1.max()

A função `max()` toma qualquer número de argumentos e retorna o maior deles. ("Maior" pode ter definições estranhas, então é melhor usar `max()` em números inteiros e de ponto flutuante, onde os resultados são simples, não em outros objetos, como strings.)

Por exemplo, `max(1,2,3)` retornará `3`(o maior número no conjunto de argumentos).

Instruções

Experimente a função `max()` na linha 3 do editor. Você pode fornecer qualquer número de argumentos inteiros ou de ponto flutuante a `max()`.

8.12.2.min()

`min()` retorna então o menor de uma dada série de argumentos.

Instruções

Vá em frente e iguale `minimum` ao `min()` de qualquer conjunto de números inteiros ou de ponto flutuante que você quiser.

8.12.3.abs()

A função `abs()` retorna o módulo do número que toma como argumento — ou seja, a distância entre esse número e `0` em uma reta numérica imaginária. Por exemplo, `3` e `-3` têm o mesmo módulo: `3`. A função `abs()`sempre retorna um valor positivo, ao contrário de `max()` e `min()`, ele toma apenas um único número.

Instruções

Iguale `absolute` ao módulo de `-42` na linha 2.

8.12.4.type()

Finalmente, a função `type()` retorna o tipo dos dados que recebe como um argumento. Se você pedir ao Python para fazer o seguinte:

```
print type(42)
print type(4.2)
```

```
print type('spam')
```

O resultado do Python será:

```
<type 'int'>
<type 'float'>
<type 'str'>
```

Instruções

Faça o Python exibir o tipo de um `int`, a `float`, e um `str` no editor. Você pode selecionar quaisquer valores para chamar `type()`, desde que sejam um de cada.

8.13. Revisão: Funções

Ok! Vamos revisar as funções.

```
def speak(message):
    return message

    if happy():
        speak("Estou feliz!")
    elif sad():
        speak("Estou triste.")
    else:
        speak("Nao sei o que estou sentindo.")
```

Novamente, o exemplo acima está aqui apenas para referência!

Instruções

1. Primeiro, defina (`def`) uma função, `shut_down`, que toma um argumento `s`. Não se esqueça dos parênteses ou dos dois pontos!
2. Então, se (`if`) a função `shut_down` recebe um `s` igual a `"yes"` ("sim"), ela deve retornar `"Desligando"`
3. Como alternativa, `elif s` é igual a `"no"`, então a função deve retornar `"Desligamento abortado"`.
4. Finalmente, se `shut_down` receber qualquer coisa além dessas entradas, a função de retornar `return "Desculpe"`

8.14. Revisão: Módulos

Bom trabalho! Agora vamos ver o que você se lembra sobre importação de módulos (e, especificamente, o que está disponível no módulo `math`).

Instruções

Importe o módulo `math` do modo que achar melhor. Chame a função `sqrt` para o número `13689` e exiba (`print`) esse valor no console.

8.15. Revisão: Funções Embutidas

Perfeito! Por último, mas não menos importante, vamos revisar as funções embutidas que você aprendeu nesta lição.

```
def is_numeric(num):
    return type(num) == int or type(num) == float

max(2, 3, 4) # 4
min(2, 3, 4) # 2

abs(2) # 2
abs(-2) # 2
```

Instruções

1. Primeiro, defina (`def`) uma função chamada `distance_from_zero`, com um argumento (escolha qualquer nome de argumento que desejar).
2. Se o tipo (`type`) do argumento for `int` ou `float`, a função dever retornar (`return`) o módulo, ou valor `abs`oluto da função de entrada.
3. Caso contrário, a função deve retornar "Nao"

9. ATIVIDADE: TIRANDO FÉRIAS

9.1. Antes de Começar

Primeiro vamos revisar rapidamente as funções no Python.

```
def bigger(first, second):
    print max(first, second)
    return True
```

no exemplo acima:

1. Definimos uma função chamada `bigger` que tem dois argumentos chamados `first` e `second`.
2. Então, exibimos o maior dos dois argumentos usando a função embutida `max`.
3. Finalmente, a função `bigger` retorna `True`.

   Agora tente você mesmo criar uma função!

Instruções

Escreva uma função chamada `answer` que não tome nenhum argumento e retorne o valor `42`.

Mesmo sem argumentos, você precisará dos parênteses.

Não se esqueça dos dois pontos no fim da definição da função!

9.2. Planejando Sua Viagem

Quando se planejam férias, é muito importante saber exatamente quanto você vai gastar.

```
def wages(hours):
    # Se eu ganho $8.35/hora...
    return 8.35 * hours
```

O exemplo acima e apenas um lembrete sobre como as funções são definidas.

Vamos usar funções para calcular o custo da sua viagem.

Instruções

1. Defina uma função chamada `hotel_cost` com um argumento `nights` como entrada.
2. O hotel custa $140 por noite. Então, a função `hotel_cost` deve `retornar 140 * nights`.

9.3. Chegando Lá

Você vai precisar viajar de avião para chegar ao seu destino.

```
def fruit_color(fruit):
    if fruit == "apple":
        return "red"
    elif fruit == "banana":
        return "yellow"
    elif fruit == "pear":
        return "green"
```

1. O exemplo acima define a função `fruit_color` que aceita uma string como argumento `fruit`.
2. A função retorna uma string se souber qual é a cor da fruta (`fruit`).

Instruções

1. Abaixo do seu código já existente, defina uma função chamada `plane_ride_cost` que toma uma string, `city`, como entrada.
2. A função deve `retornar` um preço diferente dependendo do local, similar ao exemplo acima. Abaixo são mostrados os destinos válidos e seus preços correspondentes para viagens de ida e volta.

```
"Charlotte":183
"Tampa":220
"Pittsburgh":222
"Los Angeles":475
```

9.4. Transporte

Você também vai precisar alugar um carro para poder se locomover.

```
def finish_game(score):
    tickets = 10 * score
    if score >= 10:
        tickets += 50
    elif score >= 7:
        tickets += 20
    return tickets
```

No exemplo acima, primeiro damos ao jogador 10 tíquetes para cada ponto que o jogador marcar. Então, verificamos o valor de `score` diversas vezes.

1. Primeiro, verificamos se `score` é maior ou igual a 10. Se for, damos ao jogador 50 tíquetes adicionais.
2. Se `score` for apenas maior ou igual a 7, damos ao jogador 20 tíquetes adicionais.
3. No fim, retornamos o número total de tíquetes ganhados pelo jogador.

Lembre-se que uma declaração `elif`e verificada apenas se todas as declarações `if/elif` anteriores falharem.

Instruções

1. Abaixo do seu código já existente, defina uma função chamada `rental_car_cost` com um argumento chamado `days`.
2. Calcule o custo de alugar um carro:
   - O aluguel do carro custa R$40 por dia.
   - Se (`if`) você alugar o carro por 7 ou mais dias, terá um desconto de R$50 sobre o total.
   - Alternativamente (`elif`), se você alugar o carro por 3 ou mais dias, terá um desconto de R$20 sobre o total.
   - Você não pode receber os dois descontos acima.
3. Retorne o custo.

Como no exemplo acima, essa verificação fica mais simples se você fizer a verificação de 7 dias com uma declaração `if` e a verificação de 3 dias como uma declaração `elif`.

9.5. Reunindo Tudo

Grande! Agora que você descobriu os 3 custos principais, vamos reuní-los para encontrar o custo total de sua viagem.

```
def double(n):
    return 2 * n
def triple(p):
    return 3 * p

def add(a, b):
    return double(a) + triple(b)
```

1. Definimos duas funções simples, `double(n)` e `triple(p)` que retornam 2 ou 3 vezes suas entradas. Note que seus argumentos são `n` e `p`.

2. Definimos uma terceira função, `add(a, b)` que retorna a soma das duas funções anteriores quando chamadas com `a` e `b`, respectivamente.

Instruções

1. Abaixo do seu código já existente, defina uma função chamada `trip_cost`, que toma dois argumentos, `city` e `days`.
2. Como no exemplo acima, faça sua função retornar a soma da chamada das funções `rental_car_cost(days)`, `hotel_cost(days)`, e `plane_ride_cost(city)`.

É completamente válido chamar a função `hotel_cost(nights)` com a variável `days`. Como no exemplo acima em que chamamos `double(n)`com a variável `a`, passamos o valor de `days` para a nova função no argumento `nights`.

9.6. Ei, Nunca Se Sabe!

Você não pode esperar gastar dinheiro apenas com a passagem de avião e aluguel de carro quando estiver viajando. Também é preciso haver espaço para gastos adicionais, como alimentação e souvenirs.

Instruções

1. Mude a definição da sua função `trip_cost`. Adicione um terceiro argumento, `spending_money`.
2. Mude o que a função `trip_cost` faz. Adicione a variável `spending_money` à soma que ela retorna.

9.7. Planeje Sua Viagem!

Bom trabalho! Agora que você reuniu tudo, vamos fazer uma viagem.

E se quiséssemos ir para Los Angeles por 5 dias e levar 600 reais adicionais de dinheiro para despesas?

Instruções

Depois do código anterior, exiba (`print`) `trip_cost(`para `"Los Angeles"` por `5` dias com `600` reais adicionais de dinheiro para despesas.

Não se esqueça do `)` final depois de inserir os 3 valores anteriores!

```
def hotel_cost(nights):
    return 140 * nights


def plane_ride_cost(city):
    if city == "Charlotte":
        return 183
    elif city == "Tampa":
        return 220
    elif city == "Pittsburgh":
        return 222
    elif city == "Los Angeles":
        return 475


def rental_car_cost(days):
    aluguel = 40 * days
    if days >= 7:
        aluguel -= 50
    elif days >= 3:
        aluguel -= 20
    return aluguel


def trip_cost(city,days,spending_money):
    return hotel_cost(days) + plane_ride_cost(city) + rental_car_cost(days) + spending_money
```

```
print trip_cost("Los Angeles", 5, 600)
```

10. LISTAS E DICIONÁRIOS EM PYTHON

10.1. Introdução às Listas

Listas são um datatype (tipo de dado) que você pode usar para armazenar uma coleção de diferentes informações como uma sequência sob um único nome de variável. (alguns datatypes que você já conhece incluem strings, números e booleanos).

Você pode atribuir itens a uma lista com uma expressão na forma

```
list_name = [item_1, item_2]
```

com os itens entre colchetes. Uma lista também pode estar vazia: `empty_list = []`.

Listas são muito similares a strings, mas há algumas diferenças essenciais.

Instruções

A lista zoo_animals" tem três itens (veja-os na linha 1). Vá em frente e adicione um quarto! Simplesmente insira o nome do seu animal favorito (como uma `"string"`) na linha 1, depois da vírgula final, mas antes do fechamento "]".

```
zoo_animals = ["pangolin", "cassowary", "sloth",_____];

# One animal is missing!



if len(zoo_animals) > 3:

        print "O primeiro animal do zoo e " + zoo_animals[0]

        print "O segundo animal do zoo e " + zoo_animals[1]

        print "O terceiro animal do zoo e " + zoo_animals[2]
```

10.2. Acesso por Index

Você pode acessar um item individual na lista por seu índice . Um índice é como um endereço que identifica o lugar do item na lista. O índice aparece logo após o nome da lista, entre parênteses, como este: `list_name[index]`.

Índices Lista começar com 0, não 1! Você acessa o primeiro item de uma lista como esta: `list_name[0]`. O segundo item de uma lista é no índice 1: `list_name[1]`. Os cientistas da computação gostam de começar a contar a partir de zero.

Instruções

Escrever uma declaração de que imprime o resultado de adicionar o segundo e quarto itens da lista. Certifique-se para acessar a lista pelo índice!

```
numbers = [5, 6, 7, 8]



print "Adding the numbers at indices 0 and 2..."

print numbers[0] + numbers[2]

print "Adding the numbers at indices 1 and 3..."

# Your code here!
```

10.3. Novos Vizinhos

Um índice de lista comporta-se como qualquer outro nome de variável! Ele pode ser usado para acessar, bem como valores atribuir.

Você viu como acessar um índice de lista como esta:

```
zoo_animals[0]  # Gets the value "pangolin"
```

Você pode ver como atribuição trabalha na linha 5:

```
zoo_animals[2] = "hyena"  # Changes "sloth" to "hyena"
```

Instruções

Escrever uma declaração de atribuição que irá substituir o item que atualmente detém o valor `"tiger"` com um outro animal (como uma string). Pode ser qualquer animal que você gosta.

```
zoo_animals = ["pangolin", "cassowary", "sloth", "tiger"]

# Last night our zoo's sloth brutally attacked

# the poor tiger and ate it whole.



# The ferocious sloth has been replaced by a friendly hyena.
```

```
zoo_animals[2] = "hyena"



# What shall fill the void left by our dear departed tiger?

# Your code here!
```

10.4. Comprimento de lista

A lista não tem que ter um comprimento fixo. Você pode adicionar itens ao final de uma lista a qualquer momento que você gosta!

```
letters = ['a', 'b', 'c']

letters.append('d')

print(len(letters))

print(letters)
```

1. No exemplo acima, primeiro criamos uma lista chamada `letters`.
2. Então, nós adicionamos a string `'d'` ao fim da lista `letters`
3. Em seguida, vamos imprimir `4`, o comprimento de lista `letters`
4. Finalmente, imprimir `['a', 'b', 'c', 'd']`.

Instruções

Nas linhas 5, 6 e 7, acrescentar mais três itens à lista `suitcase`, assim como a segunda linha do exemplo acima. (Talvez trazer um maiô?)

Em seguida, defina `list_length` igual ao comprimento da lista `suitcase`.

```
suitcase = []

suitcase.append("sunglasses")



# Your code here!
```

```
list_length = _________ # Set this to the length of suitcase


print "There are %d items in the suitcase." % (list_length)

print suitcase
```

10.5. Fatiamento de Lista

Às vezes, você só quer acessar uma parte de uma lista.

```
letters = ['a', 'b', 'c', 'd', 'e']

slice = letters[1:3]

print(slice)

print(letters)
```

1. No exemplo acima, primeiro criamos uma lista chamada ***letters***.
2. Então, vamos dar uma subseção e armazená-lo na lista ***slice***. Começamos com o índice antes dos dois pontos e continuar até , mas não incluindo o índice após os dois pontos.
3. Em seguida, vamos imprimir ***['b', 'c']***. Lembre-se que comece a contar índices entre 0 e que paramos antes índice 3.
4. Finalmente, imprimir ***['a', 'b', 'c', 'd', 'e']***, só para mostrar que não modificou a lista original ***letters***.

Instruções

Na linha 7, criar uma lista de chamada ***middle*** contendo apenas os dois itens do meio de ***suitcase***.

Na linha 10, criar uma lista chamada ***last*** feita só os dois últimos itens de ***suitcase***.

```
suitcase = ["sunglasses", "hat", "passport", "laptop", "suit", "shoes"]



# The first and second items (index zero and one)

first = suitcase[0:2]



# Third and fourth items (index two and three)

middle =
```

```
# The last two items (index four and five)

last =
```

10.6. Cortar listas e strings

Você pode cortar uma corda exatamente como uma lista! Na verdade, você pode pensar em cordas como listas de caracteres: cada personagem é um item sequencial na lista, a partir de índice 0.

```
my_list[:2]  # Grabs the first two items
my_list[3:]  # Grabs the fourth through last items
```

Se lista de sua fatia inclui o primeiro ou o último item em uma lista (ou uma string), o índice para esse item não tem de ser incluído.

Instruções

Atribuir a ***dog*** uma fatia de ***animals*** de índice 3 até, mas não incluindo o índice 6.

Atribuir a ***frog*** uma fatia de ***animals*** de índice 6 até ao fim da cadeia.

```
animals = "catdogfrog"

# The first three characters of animals

cat = animals[:3]

# The fourth through sixth characters

dog =

# From the seventh character to the end

frog =
```

10.7. Mantendo a Ordem

Às vezes você precisa procurar um item em uma lista.

```
animals = ["ant", "bat", "cat"]
```

```
print animals.index("bat")
```

1. Primeiro, criamos uma lista chamada ***animals*** com três itens.
2. Então, nós imprimimos o primeiro índice que contém a item ***"bat"***, que vai imprimir ***1***.

Podemos também inserir itens em uma lista.

```
animals.insert(1, "dog")

print(animals)
```

1. Inserimos ***"dog"*** no índice 1, que se move tudo para baixo por 1.
2. Nós imprimir ***["ant", "dog", "bat", "cat"]***

Instruções

Utilizar ***.index(item)*** função que encontra o índice ***"duck"***. Atribuir esse resultado a uma variável chamada ***duck_index***.

Em seguida, ***.insert(index, item)*** ao item ***"cobra"*** nesse índice.

```
animals = ["aardvark", "badger", "duck", "emu", "fennec fox"]

duck_index = # Use index() to find "duck"



# Your code here!




print animals # Observe what prints after the insert operation
```

10.8. Laço For

Se você quiser fazer alguma coisa com cada item na lista, você pode usar um laço for.

```
for variable in list_name:
        # Do stuff!
```

Um nome de variável segue a forpalavra-chave; será atribuído o valor de cada item da lista, por sua vez.

Então “in list_name” designa “list_name” como a lista do laço que será trabalhado. A linha termina com dois pontos (:) e o código recuado que segue o mesmo será executado uma vez por item na lista.

Intruções

Escrever uma declaração na parte recuada do “for” que imprime um número igual ao dobro do valor de cada item na lista.

```
my_list = [1,9,3,8,5,7]

for number in my_list:

 # Your code here
```

10.9. Mais com 'for'

Se a sua lista é uma bagunça confusa, você pode precisar da função sort().

```
animals = ["cat", "ant", "bat"]

animals.sort()

for animal in animals:

        print animal
```

1. Primeiro, criamos uma lista chamada “animals” com três itens. Os itens não estão em ordem alfabética.
2. Então, nós classificamos “animals” em ordem alfabética. Note que .sort() modifica a lista em vez de retornar uma nova lista.
3. Então, para cada item de “animals”, vamos imprimir esse item como "ant", "bat", "cat" na sua própria linha de cada um.

Instruções

Escrever um “for” que itera sobre start_liste e .append() cada número ao quadrado (x ** 2) para square_list.

Em seguida, classifique square_list!

```
start_list = [5, 3, 1, 2, 4]

square_list = []

# Your code here!
```

```
print square_list
```

10.10. Esta próxima parte é chave

Um dicionário é semelhante a uma lista, mas você acessar valores, observando-se uma chave em vez de um índice. A chave pode ser qualquer cadeia ou número. Os dicionários são entre chaves, assim:

```
d = {'key1' : 1, 'key2' : 2, 'key3' : 3}
```

Este é um dicionário chamado “d” com três pares chave-valor . Os principais: 'key1' para o valor 1, 'key2' para 2, e assim por diante.
Os dicionários são grandes para coisas como livros de telefone (emparelhamento um nome com um número de telefone), páginas de login (emparelhamento um endereço de e-mail com um nome de usuário), e muito mais!

Instruções

Imprimir os valores armazenados sob as chaves 'Sloth' e 'Burmese Python'. Acessando valores de dicionário por chave é como acessar valores de lista pelo índice:

```
residents['Puffin'] # Gets the value 104
```

```
# Assigning a dictionary with three key-value pairs to residents:

residents = {'Puffin' : 104, 'Sloth' : 105, 'Burmese Python' : 106}



print residents['Puffin'] # Prints Puffin's room number



# Your code here!
```

10.11. Novas Entradas

Como listas, dicionários são mutáveis. Isso significa que eles podem ser alterados depois de serem criadas. Uma vantagem disso é que podemos adicionar novos “pares de chave/valor” para o dicionário depois de ser criado assim:

```
dict_name[new_key] = new_value
```

Um par vazio de chaves `{}` é um dicionário vazio, assim como um par vazio de `[]` é uma lista vazia.

O comprimento `len()` de um dicionário é o número de pares chave-valor que tem. Cada par conta apenas uma vez, mesmo que o valor é uma lista. (É isso mesmo: você pode colocar listas dentro de dicionários!)

Instruções

Adicione pelo menos mais três pares chave-valor para a variável `menu`, com o nome de prato (como `"string"`) para a chave e o preço (um float ou inteiro) como o valor. Aqui está um exemplo:

```
menu['Spam'] = 2.50
```

```
menu = {} # Empty dictionary

menu['Chicken Alfredo'] = 14.50 # Adding new key-value pair

print menu['Chicken Alfredo']



# Your code here: Add some dish-price pairs to menu!




print "There are " + str(len(menu)) + " items on the menu."

print menu
```

10.12. Alterando sua mente

Porque dicionários são mutáveis, eles podem ser mudados de muitas maneiras. Os itens podem ser removidos de um dicionário com o `del` comando:

```
del(dict_name[key_name])
```

Que irá remover a chave `key_name` e seu valor associado do dicionário.

Um novo valor pode ser associado com uma chave, atribuindo um valor à chave, assim:

```
dict_name[key] = new_value
```

Instruções

Excluir o `'Sloth'` e `'Bengal Tiger'` de `zoo_animals` usando del.

Defina o valor associado `'Rockhopper Penguin'` a qualquer coisa que não seja `'Arctic Exhibit'`.

```
# key - animal_name : value - location

zoo_animals = {

  'Unicorn' : 'Cotton Candy House',

  'Sloth' : 'Rainforest Exhibit',

  'Bengal Tiger' : 'Jungle House',

  'Atlantic Puffin' : 'Arctic Exhibit',

  'Rockhopper Penguin' : 'Arctic Exhibit'

}

# A dictionary (or list) declaration may break across multiple lines



# Removing the 'Unicorn' entry. (Unicorns are incredibly expensive.)

del zoo_animals['Unicorn']



# Your code here!







print zoo_animals
```

10.13. Retirando algumas coisas

Às vezes você precisa remover algo de uma lista.

```
beatles = ["john","paul","george","ringo","stuart"] beatles.remove("stuart") print beatles
```

Este código irá imprimir:

```
["john","paul","george","ringo"]
```

1. Nós criamos uma lista chamada `beatles` com 5 itens.
2. Então, nós removemos o primeiro item beatles que corresponde à cadeia `"stuart"`. Note-se que `.remove(item)` não retorna nada.
3. Finalmente, imprimir essa lista só para ver que `"stuart"` foi realmente removido.

Instruções

Remover `'dagger'` da lista de itens armazenados na variável `backpack`.

```
backpack = ['xylophone', 'dagger', 'tent', 'bread loaf']

# Your code here!
```

10.14. É perigoso ir sozinho! Pegue isso

Vamos passar por cima de algumas últimas notas sobre dicionários:

```
my_dict = { "fish": ["c", "a", "r", "p"], "cash": -4483, "luck": "good" }

print(my_dict["fish"][0])
```

No exemplo acima, criamos um dicionário que contém muitos tipos de valores.
A chave `"fish"` tem uma lista , a chave `"cash"` tem um int , e a chave `"luck"` tem uma string.
Finalmente, imprimir a string `"c"`. Quando acessar um valor no dicionário como `my_dict["fish"]`, temos acesso direto a esse valor (que passa a ser uma lista). Podemos acessar o item no índice `0` na lista armazenada pela chave `"fish"`.

Instruções

Adicionar uma chave para `inventory` chamada `'pocket'`

Defina o valor de `'pocket'` ser uma lista que consiste nas strings `'seashell'`, `'strange berry'` e `'lint'`

`.sort()` os itens na lista armazenados sob a chave `'backpack'`.

Em seguida, `.remove('dagger')` a partir da lista de itens armazenados sob a chave `'backpack'`.

Adicionar 50 para o número armazenado sob a chave `'gold'`.

```
inventory = {

  'gold' : 500,

  'pouch' : ['flint', 'twine', 'gemstone'], # Assigned a new list to 'pouch' key

  'backpack' : ['xylophone','dagger', 'bedroll','bread loaf']

}

# Adding a key 'burlap bag' and assigning a list to it

inventory['burlap bag'] = ['apple', 'small ruby', 'three-toed sloth']

# Sorting the list found under the key 'pouch'

inventory['pouch'].sort()



# Your code here
```

11. ATIVIDADE: UM DIA NO SUPERMERCADO

11.1. Antes de seguirmos à diante

Antes de começarmos o nosso exercício, devemos repassar o laço for mais uma vez. Por enquanto, vamos apenas explicar o for em termos de como ele se relaciona com listas e dicionários.
O laço for nos permite percorrer todos os elementos em uma lista do mais a esquerda para o mais à direita. Um circuito de amostra seria estruturado da seguinte forma:

```
a = ["List", "of", "some", "sort"]
for x in a:
        # Do something for every x
```

Este ciclo irá executar todo o código no bloco recuado sob o `for x in a:` comunicado. O item da lista que está atualmente a ser avaliado será `x`. Então, executando o seguinte:

```
for item in [1, 3, 21]:
        print(item)
```

Iria imprimir `1` e, em seguida `3`, e depois `21`. A variável entre `for` e `in` pode ser configurado para qualquer nome de variável (atualmente `item`), mas você deve ter cuidado para evitar o uso da palavra `list` como uma variável, uma vez que é uma palavra reservada (isto é, que significa algo especial) na linguagem Python.

Instruções

Use um `for` para imprimir todos os elementos da lista `names`.

```
names = ["Adam","Alex","Mariah","Martine","Columbus"]

# Your code here
```

11.2. Esta é a chave!

Você também pode usar um `for` em um dicionário para fazer um loop através de suas chaves com o seguinte:

```
# A simple dictionary d = {"foo" : "bar"}
for key in d:
        print(d[key])  # prints "bar"
```

Note-se que os dicionários são desordenados, o que significa que a qualquer momento você percorrer um dicionário, você vai passar por cada chave, mas você não está garantido para obtê-los em qualquer ordem particular.

Instruções

Use um `for` para percorrer o dicionário `webster` e imprimir todas as definições.

```
webster = {
  "Aardvark" : "A star of a popular children's cartoon show.",
  "Baa" : "The sound a goat makes.",
  "Carpet": "Goes on the floor.",
  "Dab": "A small amount."
}
# Add your code below!
```

11.3. Controle de fluxos e laços

Os blocos de código em um `for` pode ser tão grande ou tão pequeno como eles precisam ser.

Enquanto looping, você pode querer executar ações diferentes, dependendo do item específico na lista.

```
numbers = [1, 3, 4, 7]
for number in numbers:
        if number > 6:
                print number
        print "We printed 7."
```

1. No exemplo acima, criamos uma lista com 4 números nele.
2. Em seguida, percorrer a `numbers` lista e armazenar cada item na lista na variável `number`.
3. Em cada ciclo, se `number` for superior a 6, vamos imprimir 7.
4. Finalmente, imprimir uma frase.

Certifique-se de acompanhar o seu recuo ou você pode ficar confuso!

Instruções

Como passo 2 acima, percorrer cada item na lista de chamada `a`.

Como passo 3 acima, if o número é par, imprima-o. Você pode testar `if` o `item % 2 == 0` para ajudar.

```
a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

# Escreva seu código aqui
```

11.4. Listas + Funções

Funções também pode ter listas como entradas e realizar várias operações nessas listas.

```
def count_small(numbers):
        total = 0
        for n in numbers:
                if n < 10:
                        total = total + 1
        return total


lotto = [4, 8, 15, 16, 23, 42]
small = count_small(lotto)
print small
```

No exemplo acima, nós definimos uma função `count_small` que tem um parâmetro, `numbers`.
Nós inicializar uma variável `total` que podemos usar no `for`.
Para cada item `n` em `numbers`, se `n` for inferior a 10, incrementamos `total`.
Após o `for`, retorna-se `total`.
Após a definição da função, criamos uma matriz de números chamados `lotto`.
Nós chamamos a `count_small` função, passar `lotto`, e armazenar o resultado retornado em `small`.
Finalmente, imprimir o resultado retornado, o que é 2 uma vez que apenas 4 e 8 são menos de 10.

Instruções

Escreva uma função que conta quantas vezes a string `"fizz"` aparece em uma lista.

Escreva uma função chamada `fizz_count` que leva uma lista `x` como entrada.

Criar uma variável `count` para manter a contagem em curso. Inicialize-o a zero.

`For` cada `item in x:`, `if` esse item é igual à cadeia `"fizz"`, em seguida, incrementar o `count`.

Após o loop, agradar `return` a `count`.

Por exemplo, `fizz_count(["fizz","cat","fizz"])` deve retornar 2.

Escreva sua função

11.5. Laços em strings

Vamos executar o código do quadro e ver sua saída.

```
for letra in "Resenha":
  print letra


word = "O rato roeu a roupa do rei de Roma!"


for letra in frase:
  # só printa a letra r
  if letra == "r":
    print letra
```

11.6. Sua própria Loja!

Parabéns! Você agora é o orgulhoso proprietário de seu próprio supermercado marca “Resenha”.

```
animal_counts = { "ant": 3, "bear": 6, "crow": 2 }
```

No exemplo acima, criamos um novo dicionário chamado `animal_counts` com três entradas. Uma das entradas tem a chave `"ant"` e o valor 3.

Instruções

Criar um novo dicionário chamado ***precos*** usando ***{}*** formato como no exemplo acima.

Colocar esses valores no seu dicionário ***precos***, entre o ***{}***:

```
"banana": 4,
"apple": 2,
"orange": 1.5,
"pear": 3
```

11.7. Investindo em estoque

Bom trabalho! Como um gerente de loja, você também é responsável por manter o controle de seu estoque/inventário.

Instruções

Criar um dicionário “estoque” com os valores abaixo.

```
"banana": 6,
"apple": 2,
"orange": 32,
"pear": 15
```

11.8. Mantendo o controle da produção

Agora que você tem todas as suas informações do produto, você deve imprimir todas as suas informações de inventário.

```
once = {'a': 1, 'b': 2}
twice = {'a': 2, 'b': 4}
for key in once:
        print "Once: %s" % once[key]
        print "Twice: %s" % twice[key]
```

1. No exemplo acima, criamos dois dicionários, `once` e `twice`, que têm as mesmas teclas.
2. Porque sabemos que eles têm as mesmas chaves, que pode percorrer um dicionário e `print` valores de ambos `once` e `twice`.

Instruções

Loop através de cada tecla no `precos`.

Como no exemplo acima, para cada tecla, imprima a chave junto com seu preço e informações sobre ações. Imprimir a resposta no seguinte formato:

```
apple

precos: 2

estoque: 0
```

Como no exemplo acima, você sabe que o `precos` e `estoque` tem as mesmas chaves, você pode acessar `estoque` enquanto você está no mesmo loop que contém `precos`.

Quando você estiver imprimindo, você pode usar a sintaxe do exemplo acima.

11.9. Algo de valor

Para fins de papelada e de contabilidade, vamos gravar o valor total de seu inventário. É bom saber o que você vale a pena!

Instruções

Vamos determinar quanto dinheiro você faria se você vendeu todos os seus alimentos.

Criar uma variável chamada `total` e configurá-lo para zero.

Crie um laço através de `precos`.

Para cada chave em `precos`, multiplique o número em `precos` pelo número no `estoque`. Imprimir esse valor no console e, em seguida, adicioná-lo a `total`.

Finalmente, fora do seu loop, `print total`.

11.10. Compras no Mercado

Ótimo trabalho! Agora vamos dar um passo para trás de lado a gestão e vejam através dos olhos do cliente.

Para que os clientes para encomendar on-line, vamos ter que fazer uma interface consumidor. Não se preocupe: é mais fácil do que parece!

Instruções

Primeiro, faça uma `lista` chamada `comidas` com os valores `"banana"`, `"laranja"` e `"maça".`

11.11. Fazendo uma compra

Boa! Agora você vai precisar saber o quanto você está pagando por todos os itens da sua lista de compras.

```
def sum(numbers):
        total = 0
        for number in numbers:
                total += number
        return total
n = [1, 2, 5, 10, 13]
print sum(n)
```

1. No exemplo acima, primeiro definir uma função chamada `sum` com um parâmetro `numbers`.
2. Nós inicializamos a variável `total` que usaremos como nossa soma parcial.
3. Para cada número na lista, podemos adicionar esse número para a soma em `total`.
4. No final da função, voltamos a soma em execução.
5. Após a função, criamos, `n`, uma lista de números.
6. Finalmente, chamamos a `sum(numbers)` função com a variável ne imprimir o resultado.

Instruções

Definir uma função `computa_compra` que leva um argumento `item` como entrada.

Na função, criar uma variável total com um valor inicial de zero.

Para cada item na lista `compra`, adicione o preço desse item para total.

Finalmente, retorna total.

11.12. Gerindo o estoque

Agora você precisa que sua função `computa_compra` leve o estoque/inventário de um item em particular em consideração ao calcular o custo.

Em última análise, se um item não está em estoque, então não deve ser incluído no total. Você não pode comprar ou vender o que você não tem!

Instruções

Faça as seguintes alterações na sua função `computa_compra`:

Enquanto você percorrer cada item de `compra`, adicione o preço do item para total `if` o item em `estoque` é maior do que zero.

Se o item está em `estoque` e depois de adicionar o preço para o total, subtrair um item da contagem do `estoque`.

11.13. Vamos verificar!

Perfeito! Você fez um ótimo trabalho com listas e dicionários neste projeto. Você praticou:

- Usando laços `for` com listas e dicionários
- Criando funções com laços, listas e dicionários
- Atualizando dados em resposta a mudanças no ambiente (por exemplo, diminuindo o número de bananas no `estoque` em 1 quando você vende um).

12. O ESTUDANTE VIRA PROFESSOR

12.1. Lição número um

Bem-vindo a este "Desafio". Até agora temos lhe ajudado a trabalhar em alguns projetos curtos e relativamente fáceis. Este é um desafio para lhe deixar pronto. Temos fé em você!
Vamos soltá-lo um pouco e permitir que você seja o professor de sua própria classe. Fazer um livro de notas para todos os seus alunos.

```
animal = { "name": "Mr. Fluffles", "sounds": ["meow", "purr"] }

print animal["name"]
```

O exemplo acima é apenas para lembrá-lo como criar um dicionário e, em seguida, para acessar o item armazenado pela `"cat"` chave.

Instruções

Criar três dicionários: `Rafa`, `Malu`, e `Daniel`.

Dê a cada dicionário as chaves "name", "homework", "quizzes" e "tests".

Tem a chave "nome" seja o nome do aluno (ou seja, o nome lloyd deve ser "Lloyd") e as outras chaves deve ser uma lista vazia (Vamos preencher essas listas em breve!)

12.2. Qual é o resultado?

Ótimo trabalho!

Instruções

Agora preencha o seu dicionário `rafa` com as notas apropriadas. Para poupar algum tempo, temos preenchido o resto para você.

Homework: 90.0, 97.0, 75.0, 92.0

Quizzes: 88.0, 40.0, 94.0

Tests: 75.0, 90.0

Certifique-se de incluir os pontos decimais para suas notas! Isto será importante mais tarde.

rafa = {

```
  "name": "Rafa",
  "homework": [],
  "quizzes": [],
  "tests": []
}
alice = {
  "name": "Alice",
  "homework": [100.0, 92.0, 98.0, 100.0],
  "quizzes": [82.0, 83.0, 91.0],
  "tests": [89.0, 97.0]
}
daniel = {
  "name": "Daniel",
  "homework": [0.0, 87.0, 75.0, 22.0],
  "quizzes": [0.0, 75.0, 78.0],
  "tests": [100.0, 100.0]
}
```

12.3. União de dicionários

Agora vamos colocar os três dicionários em uma lista juntos.

```
my_list = [1, 2, 3]
```

O exemplo acima é apenas um lembrete sobre como criar uma lista. Depois, `my_list` contém `1`, `2` e `3`.

Instruções

Abaixo do seu código, crie uma lista chamada `estudantes` que contém `rafa`, `alice` e `daniel`.

12.4. Rumo ao recorde!

Excelente. Agora você precisa de um documento impresso com todas as notas dos seus alunos.

```
animal_sounds = {"cat": ["meow", "purr"], "dog": ["woof", "bark"]}

print animal_sounds["cat"]
```

O exemplo acima é apenas para lembrá-lo como criar um dicionário e, em seguida, para acessar o item armazenado pela `"cat"` chave.

Instruções

"Para" cada estudante da lista "estudantes" imprima: name, homework, quizzes e tests.

12.5. Acima da média é aprovado!

Ao ensinar uma classe, é importante tomar as médias dos alunos, a fim de atribuir notas.

```
5 / 2

# 2

5.0 / 2

# 2.5

float(5) / 2

# 2.5
```

O exemplo acima é um lembrete de como a divisão trabalha em Python.

- Quando você divide um inteiro por outro inteiro, o resultado é sempre um inteiro (arredondado para baixo, se necessário).
- Quando você divide uma boia por um número inteiro, o resultado é sempre um float.
- Para dividir dois inteiros e acabar com um flutuador, primeiro você deve usar `float()` para converter um dos números inteiros a uma boia.

Instruções

. Escreva uma função `media` que leva uma lista de números e retorna a média.

- Definir uma função chamada `media` que tem argumento `numeros`.
- Dentro dessa função, chame `sum()` com a lista `numeros` como um parâmetro. Armazenar o resultado em uma variável chamada `total`.

- Como no exemplo acima, use `float()` para converter `total` e armazenar o resultado em `total`.
- Dividir `total` pelo comprimento da lista `numeros`. Utilize `len()` para calcular isso.

Sua função pode ficar assim:

```
def media(n):
    total = 0
    for i in n:
        total += float(i)
    return total/len(n)
```

12.6. Pondere a média!

Ótimo! Agora precisamos calcular média de um aluno usando médias ponderadas.

```
cost = {
       "apples": [3.5, 2.4, 2.3],
       "bananas": [1.2, 1.8],
}


return 0.9 * average(cost["apples"]) + \
0.1 * average(cost["bananas"])
```

1. No exemplo acima, criamos um dicionário chamado `cost` que inclui os custos de algumas frutas.
2. Em seguida, calculamos o custo médio de maçãs e o custo médio de bananas. Como gostamos maçãs muito mais do que nós gostamos bananas, nós ponderar o custo médio de maçãs em 90% eo custo médio de bananas em 10%.

O `\` personagem é um caractere de continuação . A seguinte linha é considerada uma continuação da linha atual.

Instruções

Escreva uma função chamada `pega_media` que leva um dicionário estudante (como `rafa`, `alice` ou `daniel`) como entrada e retorna sua média ponderada.

- Definir uma função chamada `pega_media` que tem um argumento chamado `student`.

- Faça uma variável `homework` que armazena o `average()` de `student["homework"]`.
- Repetir o passo acima para "testes" e "ensaios".
- Multiplicar os 3 médias por seus pesos e retornar a soma desses três. Os trabalhos de casa são 10%, os questionários são 30% e 60% são testes.

12.7. Enviando uma carta

Ótimo trabalho!

Agora vamos escrever uma `get_letter_grade` que recebe um número `score` como entrada e retorna um string com a nota que esse aluno deve receber.

Instruções

Definir uma nova função chamada `nota` que tem um argumento chamado `score`. Use `score` para ser um número.

Dentro de sua função, teste `score` utilizando uma cadeia de `if: / elif: / else:` declarações, assim:

- Se a pontuação é de 90 ou acima: `return "A"`
- Senão se pontuação é de 80 ou acima: `return "B"`
- Senão se pontuação é de 70 ou acima: `return "C"`
- Senão se pontuação é de 60 ou acima: `return "D"`
- Senão: `return "F"`

Finalmente, vamos testar a sua função!

Chame sua função `nota` com o resultado de `get_average(lloyd)`. Imprimir a nota de cada resultante.

12.8. Parte do Todo

Boa! Agora vamos calcular a média da classe.

Você precisa obter a média para cada aluno e, em seguida, calcular a média dessas médias.

Instruções

Definir uma função chamada `media_classe` que tem um argumento `class_list`. Você pode usar `class_list` como uma lista contendo seus três estudantes.

Primeiro, faça uma lista vazia chamada `results`.

Para cada `student` item no `class_list`, calcular `pega_media(student)` e, em seguida, chamar `results.append()` com esse resultado.

Finalmente, devolver o resultado de chamar `media()` com `results`.

Finalmente, print o resultado da chamada `get_class_average` com a sua `estudantes`. Sua lista `estudantes` deve ser `[rafa, alice, daniel]`.

Então, `print` o resultado de `get_letter_grade` para a média da classe.

13. NUMPY

13.1. Conhecendo NumPy

O NumPy é o pacote básico da linguagem Python que permite trabalhar com arranjos, vetores e matrizes de N dimensões, de uma forma comparável e com uma sintaxe semelhante ao software proprietário Matlab, mas com muito mais eficiência, e com toda a expressividade da linguagem. Provê diversas funções e operações sofisticadas, incluindo (mas não se limitando a):

- Objeto `array` para a implementação de arranjos multidimensionais
- Objeto `matrix` para o cálculo com matrizes
- Ferramentas para álgebra linear
- Transformadas de Fourier básicas
- Ferramentas sofisticadas para geração de números aleatórios

Além disso tudo, as classes criadas podem ser facilmente herdadas, permitindo a customização do comportamento (por exemplo, dos operadores típicos de adição, subtração, multiplicação, etc.). O módulo é implementado em linguagem C, o que dá uma grande velocidade às operações realizadas.

Vamos iniciar no NumPy:

```
# começaremos importando a biblioteca com o nome "np"

import numpy as np

# façamos o primeiro array

a = np.array([10, 20, 30, 40, 50])

# chamando "a" no terminal temos

array([10, 20, 30, 40, 50])

# o tipo desse array é visto chamando type()

type(a)

saída => numpy.ndarray

# agora queremos uma matriz 2x2

matriz = np.array([[1, 2], [3, 4]])

# chamando "matriz" temos

array([[1, 2],
       [3, 4]])
```

se quisermos imprimir um elemento do vetor ou matriz chamamos print() e indicamos qual elemento queremos imprimir, ou seja, para matriz[i][j] temos que i indica qual linha estamos trabalhando começando do índice 0, e j indica a coluna começando pelo índice 0 também

```
print(matriz[1][1])
```

saída => 4

quando colocamos os índices de forma negativa estamos percorrendo o vetor ou matriz do fim para o começo

```
print(matriz[-1][-1])
```

saída => 4

para acessar uma linha inteira de uma matriz façamos

```
matriz[i,:]
```

com o i indicando qual linha se quer trabalhar

para acessar uma coluna inteira façamos

```
matriz[:,j]
```

com o j indicando qual coluna se quer trabalhar

para mostrar a transposta de uma matriz chamamos a função transpose()

```
matriz.transpose()
```

saída => array([[1, 3],

[2, 4]])

para somarmos duas matriz façamos

```
m1 = np.array([[1, 2], [3, 4]])

m2 = np.array([[5, 6], [7, 8]])

m3 = m1 + m2

print(m3)
```

saída => [[6 8]

[10 12]]

```
# para subtrair façamos o mesmo procedimento

m1 = np.array([[1, 2], [3, 4]])

m2 = np.array([[5, 6], [7, 8]])

m3 = m1 - m2

print(m3)

saída => [[-4  -4]

          [-4  -4]]

# também podemos multiplicar um escalar por uma matriz

m1 = np.array([[1, 2], [3, 4]])

m2 = 3 * m1

print(m2)

saída => [[3   6]

          [9  12]]
```

para multiplicarmos matrizes façamos o mesmo procedimento, porém lembre-se que só podemos multiplicar matrizes quando o número de colunas da primeira matriz é igual ao número de linhas da segunda.

```
m1 = np.array([[1, 2], [3, 4]])

m2 = np.array([[5, 6], [7, 8]])

m3 = m1 * m2

print(m3)

saída => [[5   12]

          [21  32]]

# se quisermos somar os números dentro de um vetor chamamos a função sum()

m1 = np.array([[1, 2, 3, 4]])

m1.sum()

saída => 10
```

```
# se quisermos dividir uma matriz por outra façamos

m1 = np.array([[1, 2], [3, 4]])

m2 = np.array([[5, 6], [7, 8]])

m3 = m1 / m2

print(m3)

saída => [[0.2         0.33333333]
          [0.42857143  0.5       ]]


# se quisermos arredondar os números dentro de uma matriz fazemos

print(np.matriz.round(m1 / m2))

saída => [[0.  0.]

          [0.  0.]]

# se quisermos somar qualquer número a qualquer número dentro de uma raiz façamos

print(m1 + 3)

saída => [[4  5]

          [6  7]]
```

Obs: para subtrair, multiplicar, dividir, elevar a uma potência ou tirar alguma raiz, podemos seguir o mesmo procedimento.

```
# se quisermos o índice com o maior valor chamamos argmax()

m1.argmax()

saída => 3

# se quisermos o índice com o menor valor chamamos argmin()

m1.argmin()

saída => 0
```

13.2. Arrumando Arrays

Vamos trabalhar com o vetor a = [10, 20, 30, 40]

```
# se quisermos mostrar os números 20 e 30, respectivamente fazemos
```

print(a[1:3])

saída => [20, 30]

onde o índice 1 em a[1:3] indica o índice do número 20 na lista e o 3 indica o limite superior, ou seja, pegamos os índices 1 e 2 que são respectivamente os números 20 e 30 mostrados.

se quero saltar números dentro de uma lista faço a[: : i] onde o i indica de qual será o tamanho do salto a dar dentro do vetor

print(a[: : 2])

saída => [10, 30]

note que do número 10 pulou diretamente para o número 30 e não apresentou o 40, isso acontece pelo efeito dessa chamada.

se quiser fazer uma cópia do vetor ou matriz para manipular sem alterar o original chamamos a função copy()

b = a.copy()

onde b terá os mesmo parâmetros de a, sendo que qualquer alteração em b não implica nenhuma alteração em a.

13.3. Matrizes com Listas

podemos manipular matrizes como se fossem listas, para isso façamos

mat = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

for linha in mat:

    print(linha)

saída => [1, 2, 3]

[4, 5, 6]

[7, 8, 9]

para imprimir uma linha qualquer façamos

print(mat[i])

onde i será o índice da linha que se quer imprimir.

se quiser transformar uma lista em um array NumPy faça

a = np.array(mat)

dessa forma “a” se pega a lista “mat” em forma de array NumPy.

13.4. Criando Matrizes com NumPy

se quisermos criar uma matriz que contenha pontuações de quatro pessoas em um determinado jogo façamos, isso vale pra qualquer exemplo.

joao_pts = [10, 30, 50, 20]

pedro_pts = [30, 40, 70, 10]

maria_pts = [20, 40, 90, 30]

marcos_pts = [50, 20, 30, 60]

pontos = np.array([joao_pts, pedro_pts, maria_pts, marcos_pts])

se quisermos criar um vetor numpy de 0 a i façamos

t = np.arange(0, i)

exemplo:

h = np.arange(0, 12)

print(h)

saída => [0 1 2 3 4 5 6 7 8 9 10 11]

se quisermos transformar um vetor em uma matriz numpy de i linhas e j colunas façamos

g = np.reshape(h, (3, 4))

print(g)

saída => [[0 1 2 3]

[4 5 6 7]

[8 9 10 11]]

13.4.1. Inserindo elementos no array

para inserir elementos no array (vetor) façamos

a = np.array([1, 3, 4, 2])

np.insert(a, i, n)

onde “i” significa o índice onde se quer inserir o número e “n” é o número que se quer inserir

print(np.insert(a, 4, 10))

saída => [1 3 4 2 10]

se quisermos inserir uma linha num array 2D podemos fazer

```
a = np.array([[1, 2], [3, 4]])
np.insert(a, i, n, axis=0)
```

para acrescentar uma nova linha na matriz na posição i, com os números n, como por exemplo:

```
print(np.insert(a, 1, 10, axis=0))
saída => [[1   2]
          [10 10]
          [3   4]]
 # se quisermos inserir uma coluna num array 2D podemos fazer
a = np.array([[1, 2], [3, 4]])
np.insert(a, i, n, axis=1)
```

para acrescentar uma nova linha na matriz na posição i, com os números n, como por exemplo:

```
print(np.insert(a, 2, 7, axis=1))
saída => [[1   2   7]
          [3   4   7]]
# para somar os números das linhas façamos
print(a.sum(axis=1))
saída => [3   7]
# para somar os números das colunas façamos
print(a.sum(axis=0))
saída => [4   6]
# para anexar valores ao final de um array chamamos a função append()
a = np.array([1, 2, 3])
print(np.append(a, [4, 5, 6]))
saída => [1  2  3  4  5  6]
# para anexar linha ao final de um array 2D com valores diversificados fazemos
b = np.array([[1, 2], [3, 4]])
np.append(b, [[5, 6]], axis=0)
saída => array([[1, 2],
                [3, 4],
                [5, 6]])
```

para anexar coluna ao final de um array 2D com valores diversificados fazemos

b = np.array([[1, 2], [3, 4]])

np.append(b, [[5], [6]], axis=1)

saída => array([[1, 2, 5],

[3, 4, 6]])

13.4.2. Deletando elementos do array

para deletar linha do array 2D façamos

r = np.array([[1, 2], [3, 4], [5, 6]])

np.delete(r, i, 0)

onde "i" é a linha que se quer deletar conforme o índice da mesma e "0" significa que o eixo é o das abscissas. Exemplo:

np.delete(r, 1, 0)

saída => array([[1, 2],

[5, 6]])

para deletar coluna do array 2D façamos

r = np.array([[1, 2], [3, 4], [5, 6]])

np.delete(r, i, 1)

onde "i" é a coluna que se quer deletar conforme o índice da mesma e "1" significa que o eixo é o das coordenadas. Exemplo:

np.delete(r, 0, 1)

saída => array([[2],

[4],

[6]])

para deletar mais de uma linha do array 2D façamos

r = np.array([[1, 2], [3, 4], [5, 6], [7, 8])

np.delete(r, np.s_[i:j], 0)

onde "i" é o índice da linha inicial a ser cortada e "j" é o índice após a última linha a ser cortada, por ser limite superior, e "0" significa que o eixo é o das abscissas. Exemplo:

np.delete(r, np.s_[1:3], 0)

saída => array([[1, 2],

[7, 8]])

para deletar mais de uma coluna do array 2D façamos

r = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

np.delete(r, np.s_[i:j], 1)

onde “i” é o índice da coluna inicial a ser cortada e “j” é o índice após a última coluna a ser cortada, por ser limite superior, e “0” significa que o eixo é o das coordenadas. Exemplo:

np.delete(r, np.s_[1:3], 1)

saída => array([[1, 4],

[5, 8]])

13.4.3. Dividindo um array

para dividir um array 2D façamos

d = np.array([[1, 2, 3], [4, 5, 6]])

np.array_split(d, i, n)

onde “i” é a quantidade de partes que você quer dividir esse array e “n” pode ser 0 caso divida em linhas ou 1 caso divida em colunas. Exemplo:

np.array_split(d, 2, 0)

saída => [array([[1, 2, 3]]), array([[4, 5, 6]])]

14. REFERÊNCIAS

LUTZ, Mark. **Aprendendo Python** / Mark Lutz, David Ascher; tradução João Tortello. – 2. Ed. – Porto Alegre : Bookman, 2007. 568 p.

https://docs.python.org/3.4/tutorial/index.html

http://www.diveintopython3.net/

https://www.codecademy.com/pt-BR/learn/learn-python

https://www.datascienceacademy.com.br/public-course?courseid=python-fundamentos

https://www.datacamp.com/courses/intro-to-python-for-data-science

http://www.andersonmedeiros.com/curso-python-gratuito/

https://br.udacity.com/course/programming-foundations-with-python--ud036