Python Flying Circus Documentation

Release 23.08

Carlo E. T. Oliveira

Aug 30, 2023

CONTENTS

Aqui vamos ter uma introdução rápida de como programar jogos para Web usando Python. Na verdade vamos usar o Brython que é o Python que funciona dentro de um navegador web como o Firefox.

CHAPTER

ONE

FUNCIONALIDADES PRIMÁRIAS

Vamos nos concentrar no Python 3, em especial no Brython que é uma versão que executa dentro do navegador.

1.1 Python como uma calculadora

Python é como se fosse uma calculadora mais avançada. Voce pode operar dois números usando "+", "-", "/" ou "*" (multiplicação).

1.2 Usando a Memória

Em uma calculdora comum você pode salvar um número na memória e depois recuperar. No Python, você pode usar qualquer nome para designar a memória. Você usa "=" para guardar algo num nome que fica antes do igual. Depois usa o nome para recuperar o valor apontado.

1.3 Memória Como Uma Lista

Você também pode usar uma memória maior criando uma lista de valores. Você pode criar uma lista implícita usando [e]. Também pode ter um objeto lista criando com list(). Você pode usar no parâmetro uma lista implícita ou um objeto intervalo. O objeto intervalo é criado com range(). Se tiver só um parâmetro, significa intervalo de zero até aquele número, exclusive range(fim). Se tiver dois é o início e o fim do intervalo range(inicio, fim) Se tiver três, o último diz a razão de quanto pula na sequência range(inicio, fim, pulo)

1.4 Usando listas

As lista podem ser percorridas para se aproveitar cada item da lista. Uma maneira é usando o comando for. No for voce usa um intervalo ou uma lista e nomeia o item da vez:

for item_da_vez in nome_da_lista_ou_intervalo :
    <faz alguma coisa aqui>

Você também pode criar uma lista dinamicamente da forma implícita:

[item_da_vez for item_da_vez in nome_da_lista_ou_intervalo]

1.5 Propriedades das Listas

Cada item de uma lista pode ser acessado pelo índice de sua posição nela. Os itens guardados em uma lista podem ser acessados usando []. O número colocado entre chaves é a posição na lista começando por zero. Nos exemplos abaixo temos outras facilidades de acesso a uma lista.

1.6 Usando o Navegador para Criar Conteúdo

Brython quer dizer Browser-Python, ou seja, um Python que interage como o navegador. No **Brython** você tem objetos especiais para acessar e modificar coisas no navegador. No navegador, partes podem ser identificadas por um nome e o Brython usa este nome para interagir com esta parte. O pacote **browser** pode ser importado para uso com:

from browser import document

Por exemplo, bem aqui embaixo está uma parte que tem o nome **um_texto**. No código vamos identificar esta parte e escrever um texo ali.

1.7 Desenhando com o Brython

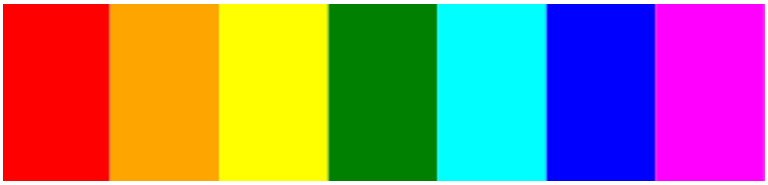
Podemos desenhar no navegador usando um padrão SVG (Scalable Vector Graphics). O pacote **browser** contem tambem o objeto svg que pode ser importado para uso com:

from browser import svg

O comando svg do pacote svg (svg.svg) cria uma tela onde você pode desenhar usando este padrão. O operador <= significa adicionar algo na tela ou mesmo adicionar um objeto dentro de outro. Vamos desenhar coisas dentro de uma parte que está bem aqui em baixo chamada **um_desenho**. Tente desenhar um boneco de palitos usando alguns dos comandos apresentados.

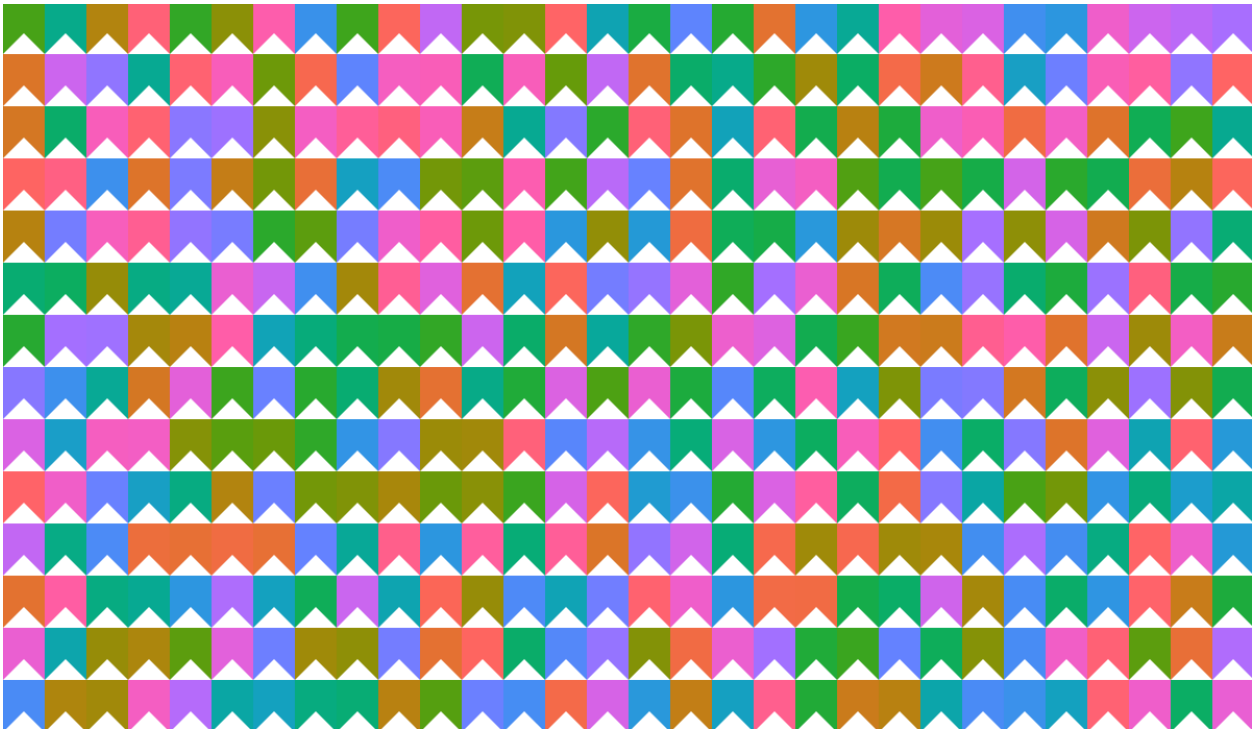
1.8 Desafio do Arco Iris

Use um comando for para desenhar as cores do arco iris com retângulos. Use a parte abaixo chamada **arco_iris**. Veja a imagem exemplo abaixo. Um desafio maior seria desenhar o arco-iris usando o comando svg.path .



1.9 Desafio do Quadro de Bandeirinhas

Alfredo Volpi foi um artista brasileiro que gostava de pintar bandeirinhas. Use um comando for para desenhar bandeirinhas de várias cores. Use a parte abaixo chamada **volpi**. Veja a imagem exemplo abaixo. Você pode usar o comando choice do pacote random para sortear uma cor diferente para cada bandeirinha.



Note: Você também pode tentar resolver algo no dojo inicial: *Problemas para o Dojo Inicial*

CHAPTER

TWO

APRENDA PYTHON EM DEZ MINUTOS I

Vamos nos concentrar no Python 3, pois essa é a versão que você deve usar. Todos os exemplos do livro estão em Python 3, e se alguém aconselhar você a usar 2, não é seu amigo.

2.1 Propriedades

Python é fortemente tipado (ou seja, os tipos são impostos), dinamicamente, implicitamente tipado (ou seja, você não precisa declarar variáveis), sensível a maiúsculas (ou seja, var e VAR são duas variáveis diferentes) e orientado a objetos (ou seja, tudo é um objeto) .

2.2 Conseguindo ajuda

A ajuda em Python está sempre disponível diretamente no interpretador. Se você quiser saber como um objeto funciona, tudo o que você precisa fazer é chamar help(<object>)! Também são úteis dir(), que mostra todos os métodos do objeto, e <object>.__doc__, que mostra sua string de documentação:

```
>>> help(5)
Ajuda no objeto int:
(etc etc)
```

```
>>> dir(5)
['__abs__', '__adicionar__', ...]
```

```
>>> abs.__doc__
'abs(número) -> número
```

Retorna o valor absoluto do argumento.

2.3 Sintaxe

O Python não possui caracteres de terminação de instrução obrigatórios e os blocos são especificados por recuo. Recuar para iniciar um bloco, recuar para terminar um. As instruções que esperam um nível de recuo terminam em dois pontos (:). Os comentários começam com o sinal de sustenido (#) e são strings de linha única e várias linhas são usadas para comentários de várias linhas. Os valores são atribuídos (na verdade, os objetos são vinculados a nomes) com o sinal de igual ("="), e o teste de igualdade é feito usando dois sinais de igual ("=="). Você pode incrementar/diminuir valores usando os operadores += e -= respectivamente pelo valor do lado direito. Isso funciona em muitos tipos de dados, incluindo strings. Você também pode usar várias variáveis em uma linha. Por exemplo:

```
>>> myvar = 3
>>> myvar += 2
>>> myvar
5
>>> myvar -= 1
>>> myvar
4
"""Este é um comentário de várias linhas.
 As linhas a seguir concatenam as duas strings."""
>>> mystring = "Hello"
>>> mystring += " world."
>>> print(mystring)
Hello world.
# Isso troca as variáveis em uma linha(!).
# Não viola a tipagem forte porque os valores não são
# realmente sendo atribuído, mas novos objetos são vinculados a
# os nomes antigos.
>>> myvar, mystring = mystring, myvar
```

See also:

Nomes e Valores em Python

2.4 Tipos de dados

As estruturas de dados disponíveis em python são listas, tuplas e dicionários. Os conjuntos estão disponíveis na biblioteca de conjuntos (mas são integrados ao Python 2.5 e posterior). Listas são como arrays unidimensionais (mas você também pode ter listas de outras listas), dicionários são arrays associativos (também conhecidos como tabelas de hash) e tuplas são arrays unidimensionais imutáveis (os "arrays" do Python podem ser de qualquer tipo, então você pode misturar, por exemplo, inteiros, strings, etc em listas/dicionários/tuplas). O índice do primeiro item em todos os tipos de array é 0. Números negativos contam do final para o início, -1 é o último item. Variáveis podem apontar para funções. O uso é o seguinte:

```
>>> sample = [1, ["another", "list"], ("a", "tuple")]
>>> mylist = ["List item 1", 2, 3.14]
>>> mylist[0] = "List item 1 again" # We're changing the item.
>>> mylist[-1] = 3.21 # Here, we refer to the last item.
>>> mydict = {"Key 1": "Value 1", 2: 3, "pi": 3.14}
>>> mydict["pi"] = 3.15 # This is how you change dictionary values.
>>> mytuple = (1, 2, 3)
>>> myfunction = len
```

(continues on next page)

(continued from previous page)

```
>>> print(myfunction(mylist))
3
```

Você pode acessar intervalos de array usando dois pontos (:). Deixar o índice inicial vazio assume o primeiro item, deixando o índice final assume o último item. A indexação é inclusiva-exclusiva, portanto, especificar [2:10] retornará os itens [2] (o terceiro item, devido à indexação 0) a [9] (o décimo item), inclusive (8 itens). Índices negativos contam do último item para trás (portanto -1 é o último item) assim:

```
>>> mylist = ["List item 1", 2, 3.14]
>>> print(mylist[:])
['List item 1', 2, 3.1400000000000001]
>>> print(mylist[0:2])
['List item 1', 2]
>>> print(mylist[-3:-1])
['List item 1', 2]
>>> print(mylist[1:])
[2, 3.14]
# Adicionando um terceiro parâmetro, "step" terá o passo do Python em
# N incrementos de itens, em vez de 1.
# Por exemplo, isso retornará o primeiro item, depois irá para o terceiro e
# retorna isso (portanto, itens 0 e 2 na indexação 0).>>> print(mylist[::2])
['List item 1', 3.14]
```

2.5 Strings

Suas strings podem usar aspas simples ou duplas, e você pode ter aspas de um tipo dentro de uma string que usa o outro tipo (ou seja, “He said ‘hello’.” é válido). Strings de várias linhas são colocadas entre _aspas duplas triplas (ou simples)_ (“””). As strings do Python são sempre Unicode, mas há outro tipo de string que é bytes puros. Esses são chamados de bytestrings e são representados com o prefixo b, por exemplo b’Hello xcexb1’. . Para preencher uma string com valores, você usa o operador % (módulo) e uma tupla. Cada %s é substituído por um item da tupla, da esquerda para a direita, e você também pode usar substituições de dicionário, assim:

```
>>> print("Nome: %s\
Número: %s\
String: %s" % (myclass.name, 3, 3 * "-"))
Nome: Stavros
Número: 3
String: ---
```

strString = “””Isso é uma string multilinha “””

AVISO: Cuidado com os s finais em “%(key)s”. >>> print(“Este %(verbo)s é um %(substantivo)s.” % {“substantivo”: “teste”, “verbo”: “é”}) Isto é um teste.

```
>>> nome = "Stavros"
>>> "Olá, {}!".format(name)
Olá, Stavros!
>>> print(f"Olá, {nome}!")
Olá, Stavros!
```

2.6 Declarações de controle de fluxo

As instruções de controle de fluxo são if, for e while. Use for para enumerar os membros de uma lista. Para obter uma lista de números, use range(<number>). A sintaxe dessas declarações é assim:

```
rangelist = range(10)
>>> print(rangelist)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

for número in rangelist:
    # Verifique se o número é um dos
    # os números na tupla.
    if número in (3, 4, 7, 9):
        # "Break" termina um for sem
        # executar a cláusula "else".
        break
    else:
        # "continue" inicia a próxima iteração
        # do laço. É bastante inútil aqui,
        # pois é a última instrução do loop.
        continue
else:
    # A cláusula "else" é opcional e é
    # executado apenas se o loop não "quebrar".
    pass # Não faça nada

if rangelist[1] == 2:
    print("O segundo item (as listas são baseadas em 0) é 2")
elif rangelist[1] == 3:
    print("O segundo item (as listas são baseadas em 0) é 3")
else:
    print("Não sei")

while rangelist[1] == 1:
    pass
```

2.7 Funções

As funções são declaradas com a palavra-chave def. Os argumentos opcionais são definidos na declaração da função após os argumentos obrigatórios, recebendo um valor padrão. Para argumentos nomeados, o nome do argumento recebe um valor. As funções podem retornar uma tupla (e usando a descompactação da tupla, você pode efetivamente retornar vários valores). As funções lambda são funções ad hoc compostas por uma única instrução. Os parâmetros são passados por referência, mas os tipos imutáveis (tuplas, ints, strings, etc) não podem ser alterados no chamador pelo chamado. Isso ocorre porque apenas o local de memória do item é passado e vincular outro objeto a uma variável descarta o antigo, portanto, os tipos imutáveis são substituídos. Por exemplo:

```
# O mesmo que def funcvar(x): return x + 1
funcvar = lambda x: x + 1
>>> print(funcvar(1))
2
```

(continues on next page)

(continued from previous page)

```
# an_int e a_string são opcionais, eles possuem valores padrão
# se um não for passado (2 e "Uma string padrão", respectivamente).
def passing_example(a_list, an_int=2, a_string="Uma string padrão"):
    a_list.append("Um novo item")
    an_int = 4
    return a_list, an_int, a_string

>>> minha_lista = [1, 2, 3]
>>> meu_int = 10
>>> print(passing_example(my_list, my_int))
([1, 2, 3, 'Um novo item'], 4, "Uma string padrão")
>>> minha_lista
[1, 2, 3, 'Um novo item']
>>> meu_int
10
```

2.8 Exercícios para este capítulo

Agora você pode tentar os exercícios em:

Problemas para o Dojo Inicial

Note: Procure ser cooperativo com a sua equipe.

CHAPTER

THREE

APRENDA PYTHON EM DEZ MINUTOS II

3.1 Classes

Python suporta uma forma limitada de herança múltipla em classes. Variáveis e métodos privados podem ser declarados (por convenção, isso não é imposto pela linguagem) adicionando pelo menos dois sublinhados iniciais e no máximo um final (por exemplo, __spam). Também podemos vincular nomes arbitrários a instâncias de classe. Segue um exemplo:

```
class MinhaClasse(object):
    comum = 10
    def __init__(self):
        self.myvariable = 3
    def minhafunção(self, arg1, arg2):
        return self.myvariable

    # Esta é a instanciação da classe

>>> instância_de_classe = MinhaClasse()
>>> instância_de_classe.minhafunção(1, 2)
3
# Esta variável é compartilhada por todas as instâncias.
>>> instância_de_classe2 = MinhaClasse()
>>> instância_de_classe2.comum
10
>>> instância_de_classe22.comum
10
# Observe como usamos o nome da classe
# em vez da instância.
>>> MinhaClasse.comum = 30
>>> instância_de_classe.comum
30
>>> instância_de_classe2.comum
30
# Isso não atualizará a variável na classe,
# em vez disso, ele vinculará um novo objeto ao antigo
# nome variável.
>>> instância_de_classe.comum = 10
>>> instância_de_classe.comum
10
>>> instância_de_classe2.comum
30
```

(continues on next page)

(continued from previous page)

```
>>> MinhaClasse.comum = 50
# Isso não mudou, porque "comum" é
# agora uma variável de instância.
>>> instância_de_classe.comum
10
>>> instância_de_classe2.comum
50

# Esta classe herda de MinhaClasse. O exemplo
# classe acima herda de "object", o que torna
# é o que chamamos de "classe de novo estilo".
# Herança múltipla é declarada como:
# class OtherClass(MinhaClasse1, MinhaClasse2, MinhaClasseN)
class OutraClasse(MinhaClasse):
    # O argumento "self" é passado automaticamente
    # e se refere à instância da classe, então você pode definir
    # variáveis de instância como acima, mas de dentro da classe.
    def __init__(self, arg1):
        self.myvariable = 3
        print(arg1)

>>> instância_de_classe = OutraClasse("Olá")
Olá
>>> instância_de_classe.minhafunção(1, 2)
3
# Esta classe não tem um membro .teste, mas
# podemos adicionar um à instância de qualquer maneira. Observação
# que este será apenas um membro da classinstance.
>>> instância_de_classe.teste = 10
>>> instância_de_classe.teste
10
```

3.2 Exceções

Exceções em Python são tratadas com blocos try-except [exceptionname]:

```
def alguma_função():
    try:
        # Divisão por zero gera uma exceção
        10/0
    except ZeroDivisionError:
        print("Opa, inválido.")
    else:
        # A exceção não ocorreu, estamos bem.
        pass
    finaly:
        # Isso é executado após o bloco de código ser executado
        # e todas as exceções foram tratadas, mesmo
        # se uma nova exceção for levantada durante o manuseio.
        print("Acabamos com isso.")
```

(continues on next page)

(continued from previous page)

```
>>> alguma_função()
Ops, inválido.
Acabamos com isso.
```

3.3 Importando

Bibliotecas externas são usadas com a palavra-chave import [libname]. Você também pode usar from [libname] import [funcname] para funções individuais. Aqui está um exemplo:

```
import random
from time import clock

randomint = random.randint(1, 100)
>>> print(randomint)
64
```

3.4 E/S de arquivo

O Python possui uma ampla variedade de bibliotecas incorporadas. Como exemplo, aqui está como a serialização (conversão de estruturas de dados em strings usando a biblioteca pickle) com E/S de arquivo é usada:

```
import pickle
minhalista = ["Isto", "é", 4, 13327]
# Abra o arquivo C:\\binary.dat para escrita. A letra r antes do
# string de nome de arquivo é usada para evitar o escape de barra invertida.
meuarquivo = open(r"C:\\binary.dat", "wb")
pickle.dump(minhalista, meuarquivo)
meuarquivo.fechar()

meuarquivo = open(r"C:\\text.txt", "w")
meuarquivo.write("Esta é uma string de amostra")
meuarquivo.close()

meuarquivo = open(r"C:\\text.txt")
>>> print(meuarquivo.read())
'Esta é uma string de amostra'
meuarquivo.close()

# Abra o arquivo para leitura.
meuarquivo = open(r"C:\\binary.dat", "rb")
lista_carregada = pickle.load(meuarquivo)
meuarquivo.fechar()
>>> print(lista_carregada)
['Isto', 'é', 4, 13327]

# Abra o arquivo para leitura usando with.
with open(r"C:\\binary.dat", "rb") as meuarquivo:
```

(continues on next page)

(continued from previous page)

```
    lista_carregada = pickle.load(meuarquivo)
    # não precisa fechar o arquivo, ao sair do bloco o with chama o close
    print(lista_carregada)
['Isto', 'é', 4, 13327]
```

3.5 Diversos

As condições podem ser encadeadas: 1 < a < 3 verifica se a é menor que 3 e maior que 1. Você pode usar del para excluir variáveis ou itens em arrays. As compreensões de lista fornecem uma maneira poderosa de criar e manipular listas. Eles consistem em uma expressão seguida por uma cláusula for seguida por zero ou mais cláusulas if ou for, assim:

```
>>> lst1 = [1, 2, 3]
>>> lst2 = [3, 4, 5]
>>> print([x * y for x in lst1 for y in lst2])
[3, 4, 5, 6, 8, 10, 9, 12, 15]
>>> print([x for x in lst1 if 4 > x > 1])
[2, 3]
# Verifica se uma condição é verdadeira para algum item.
# "qualquer" retorna verdadeiro se algum item da lista for verdadeiro.
>>> any([i % 3 para i em [3, 3, 4, 4, 3]])
True
# Isso ocorre porque 4 % 3 = 1, e 1 é verdadeiro, então any()
# retorna True.

# Verifica quantos itens uma condição é verdadeira.
>>> sum(1 for i in [3, 3, 4, 4, 3] if i == 4)
2
>>> del lst1[0]
>>> print(lst1)
[2, 3]
>>> del lst1
```

As variáveis globais são declaradas fora das funções e podem ser lidas sem nenhuma declaração especial, mas se você quiser escrever nelas, deve declará-las no início da função com a palavra-chave global, caso contrário, o Python vinculará esse objeto a uma nova variável local (cuidado com isso, é um pequeno problema que pode te pegar se você não souber). Por exemplo:

```
número = 5

def minhafunc():
    # Isso imprimirá 5.
    print(número)

def outra função():
    # Isso gera uma exceção porque a variável não
    # foi encadernado antes da impressão. Python sabe que é um
    # objeto será vinculado a ele mais tarde e cria um novo objeto local
    # em vez de acessar o global.
    print(número)
```

(continues on next page)

(continued from previous page)

```
    número = 3

def aindaoutrafunc():
    global número
    # Isso alterará corretamente o global.
    número = 3
```

3.6 Epílogo

Este tutorial não pretende ser uma lista exaustiva de todos (ou mesmo um subconjunto) do Python. Python tem uma vasta gama de bibliotecas e muito mais funcionalidades que você terá que descobrir por outros meios, como o excelente livro Dive into Python. Espero ter facilitado sua transição no Python. Por favor, deixe comentários se você acredita que há algo que poderia ser melhorado ou adicionado ou se há algo que você gostaria de ver (aulas, tratamento de erros, qualquer coisa).

Note: Procure ser cooperativo com a sua equipe.

CHAPTER

FOUR

NOMES E VALORES EM PYTHON

Um conceito elementar em programação é armazenar e lidar com valores que representam tudo que se quer computar. Em um programa Python estes valores podem ser de diversos tipos e para lidar com eles colocamos nomes para melhor lembrar o que representam.

```
tom = "um gato" # aqui o nome tom foi colocado para identificar o texto -um gato-
jerry = "um rato" # aqui jerry indentifica o texto -um rato-
num_bichos = 2 # onome num_bichos identifica o valor 2
print("O desenho tem ", num_bichos, 'personagens: ' tom, "e ", jerry)
# O texto impresso será: -O desenho tem 2 personagens: um gato e um rato-
```

A expressão *tom = "um gato"* quer dizer: use o nome *tom* para se referenciar ao texto *um gato*. Diferentemente do uso em matemática o símbolo = não quer dizer igualdade. No Python, o símbolo igual (=) quer dizer: atribua ao nome *tom* o valor *um gato*. Em todo lugar que você colocar o nome tom, o Python vai substituir pelo seu valor *um gato*. Funciona como na comunicação internet, onde você usa o emoji e o leitor entende que você na verdade escreveu: *isto é engraçado*. A expressão <*um nome*> = <*um valor*> no Python é chamada de atribuição. Com ela declaramos que doravante este nome representa este valor indicado.

Note: É importante ressaltar que um nome só poderá ser usado no programa se antes ele foi atribuído a um valor.

Existe umas regras de boas maneiras para escolher e escrever um nome em Python. Estas regras foram definidas pela Python Software Foundation em um documento chamado PEP8.

Em sua maior parte os nomes devem ser escritos em letras minúsculas com as palavras separadas pelo caracter *sublinhado(_)* por exemplo: *meu_gato*. Se o nome for se referir a uma constante, isto é, se o seu valor nunca será trocado em todo o programa, então todas as letras serão em caixa alta. Por exemplo *PT_BR = "Português do Brasil"*. Caso você vá definir um tipo novo que você criou, a norma é usar o **PascalCase** ou **UpperCamelCase**, com todas as palavras grudadas e começadas em caixa alta. Por exemplo, a palavra *class* em Python é usada para criar um tipo novo: *class GatoFrajola: <continua..>*

Note: Procure ser cooperativo com a sua equipe.

CHAPTER

FIVE

TIPOS NA LINGUAGEM PYTHON

Um conceito central do Python é que todos os valores são caracterizados por um tipo. A tipagem forte significa que o tipo de um objeto não muda de forma inesperada. Uma string contendo apenas dígitos não se torna magicamente um número, como pode acontecer em linguagens de tipagem fraca como JavaScript e Perl. Cada mudança de tipo requer uma conversão de tipo explícita (aka casting).

Outra ideia interessante no Python é o que é conhecido como *duck typing* ou tipagem do pato. Este modelo de tipagem não exige uma correspondencia absoluta entre os tipos de valores para se usar as capacidades do tipo. Ou seja, podemos fazer um nome apontar para valores com tipos semelhantes e o código vai funcionar. Se nada como um pato, anda como um pato e grasna como um pato então podemos considerar que é um pato.

Os tipos da linguagem Python

```
tom = "um gato" # aqui o nome tom foi colocado para identificar o texto -um gato-
jerry = "um rato" # aqui jerry indentifica o texto -um rato-
num_bichos = 2 # onome num_bichos identifica o valor 2
print("O desenho tem ", num_bichos, 'personagens: ' tom, "e ", jerry)
# O texto impresso será: -O desenho tem 2 personagens: um gato e um rato-
```

A expressão *tom = "um gato"* quer dizer: use o nome *tom* para se referenciar ao texto *um gato*. Diferentemente do uso em matemática o símbolo = não quer dizer igualdade. No Python, o símbolo igual (=) quer dizer: atribua ao nome *tom* o valor *um gato*. Em todo lugar que você colocar o nome tom, o Python vai substituir pelo seu valor *um gato*. Funciona como na comunicação internet, onde você usa o emoji e o leitor entende que você na verdade escreveu: *isto é engraçado*. A expressão *<um nome> = <um valor>* no Python é chamada de atribuição. Com ela declaramos que doravante este nome representa este valor indicado.

Note: É importante ressaltar que um nome só poderá ser usado no programa se antes ele foi atribuído a um valor.

Existe umas regras de boas maneiras para escolher e escrever um nome em Python. Estas regras foram definidas pela Python Software Foundation em um documento chamado PEP8.

Em sua maior parte os nomes devem ser escritos em letras minúsculas com as palavras separadas pelo caracter *sublinhado(_)* por exemplo: *meu_gato*. Se o nome for se referir a uma constante, isto é, se o seu valor nunca será trocado em todo o programa, então todas as letras serão em caixa alta. Por exemplo *PT_BR = "Português do Brasil"*. Caso você vá definir um tipo novo que você criou, a norma é usar o **PascalCase** ou **UpperCamelCase**, com todas as palavras grudadas e começadas em caixa alta. Por exemplo, a palavra *class* em Python é usada para criar um tipo novo: *class GatoFrajola: <continua..>*

Note: Procure ser cooperativo com a sua equipe.

CHAPTER

SIX

APRENDENDO OPERAÇÕES

6.1 1. Tipo Booleano

A lógica booleana é uma forma de matemática onde existem apenas dois valores **Verdadeiro** (True) e **Falso** (False)

6.2 2. Comparações Lógicas

Uma forma de controlar o que o seu programa faz é o uso de operações condicionais. O comando if <*comparação*> : <*faz se verdadeiro*> else: <*faz se falso*>

6.3 3. Conectores Lógicos

As linguagens de programação, utilizam os conectivos lógicos da lógica formal, ou melhor da lógica Aristotélica, na construção de expressões lógicas. Existem 2 conectivos lógicos e, mesmo que não os conheçamos com o nome de conectivos lógicos, utilizamo-os constantemente ao conversarmos ou então, para explicarmos qualquer disciplina a outra pessoa.

Os conectivos lógicos são:

Conectivo de conjunção
E and

Conectivo de disjunção
OU or

Partícula de Negação
NÃO not

Por exemplo, a simples frase A e B são caracteres iguais implica numa expressão lógica e acabamos de representar a mesma textualmente. Porém, a expressão pode ser facilmente escrita matematicamente, ou então, com o uso de uma linguagem de programação.

6.4 4. Comparações Numéricas

Você pode comparar dois números em Python como se faz na matemática.

Os comparadores numéricos são:

Menor
: <

Menor ou Igual
: <=

Igual
: ==

Maior
: >

Maior ou Igual
: >=

Diferente
: !=

6.5 4. Expressões Numéricas

Você pode comparar três números em Python como se faz na matemática.

Os comparadores numéricos são aqueles já explicados:

Note: Agora você pode tentar os desafios de 1 a 8 do : *Problemas para o Dojo de Aquecimento*

See also:

Página de Referência na internete: OPERADORES LÓGICOS EM PYTHON

CHAPTER

SEVEN

PROBLEMAS PARA O DOJO DE AQUECIMENTO

7.1 1. Quando dormir

O parâmetro trabalho é True se for um dia de trabalho e o parâmetro férias é True se estivermos de férias. Nós dormimos se não for um dia de trabalho ou estamos de férias. Retorne True se dormirmos até tarde..

7.2 2. Macacos encrenqueiros

Temos dois macacos, a e b, e os parâmetros a_sorri e b_sorri indicam se cada um está sorrindo. Estamos em apuros se ambos estiverem sorrindo ou se nenhum deles estiver sorrindo. Retorne True se estivermos com problemas..

7.3 3. Soma ou dobro

Dados dois valores int, retorne sua soma. Caso os dois valores sejam iguais, retorne o dobro da soma.

7.4 4. Diferença para 21

Dado um int n, retorne a diferença absoluta entre n e 21, exceto retorne o dobro da diferença absoluta se n for maior que 21..

7.5 5. Papagaio Tagarela

Temos um papagaio falando alto. O parâmetro "hora" é a hora atual no intervalo 0..23. Estamos com problemas se o papagaio estiver falando e a hora for antes das 7 ou depois das 20. Retorne True se estivermos com problemas..

7.6 6. Dezena ou Soma

Dados 2 inteiros, a e b, retorne True se um deles for 10 ou se sua soma for 10.

7.7 7. Em torno de Cem

Dado um int n, retorne True se estiver à distância menor que 10 do valor 100. Nota: abs(num) calcula o valor absoluto de um número.

7.8 8. Negatividade

Dados 2 valores int, retorne True se um for negativo e um for positivo. Exceto se o parâmetro "negativo" for True, então retorne True somente se ambos forem negativos.

7.9 9. Negação

Dada uma string, retorne uma nova string onde "nega " foi adicionado à frente. No entanto, se a string já começar com "nega", retorne a string inalterada..

7.10 10. Eliminação

Dada uma string não vazia e um int n, retorne uma nova string onde o char no índice n foi removido. O valor de n será um índice válido de um char na string original (ou seja, n estará no intervalo 0..len(str)-1 inclusive).

7.11 11. Trocação

Dada uma string, retorne uma nova string onde o primeiro e a última letras foram trocadas.

7.12 12. Copiação

Dada uma string, diremos que a frente são os 3 primeiros caracteres da string. Se o comprimento da string for menor que 3, a frente é o que estiver lá. Retorna uma nova string que tem 3 cópias da frente..

CHAPTER

EIGHT

PROBLEMAS PARA O DOJO DE AQUECIMENTO 1

8.1 1. Multiplicando as palavras

Dada uma string e um int n não negativo, diremos que a frente da string são os primeiros 3 caracteres, ou o que estiver lá se a string for menor que 3. Retorna n cópias da frente;

frente_vezes('Chocolate', 2) → 'ChoCho' frente_vezes('Chocolate', 3) → 'ChoChoCho' frente_vezes('Abc', 3) → 'AbcAbcAbc'

8.2 2. Criando uma nova palavra

Dada uma string, retorne uma nova string feita de todos os caracteres alternados (um sim, um não), começando com o primeiro, então "Hello" produz "Hlo".

8.3 3. Número Repetido

Dado um array de ints, retorne a quantidade de números de 9 no array.

8.4 4. É ou não é

Dado um array de ints, retorne True se um dos primeiros 4 elementos do array for 9. O comprimento do array pode ser menor que 4.

8.5 5. Sequência de Números

Dada uma matriz de ints, retorne True se a sequência de números 1, 2, 3 aparecer na matriz em algum lugar.

8.6 6. Explosão de Letras

Dada uma string não vazia como “Code”, retorne uma string como “CCoCodCode”.

8.7 7. Posições Iguais

Dadas 2 strings, a e b, retorne o número de posições onde elas contêm a mesma substring de comprimento 2. Portanto, “xxcaazz” e “xxbaaz” resultam em 3, já que as substrings “xx”, “aa” e “az” aparecem no mesmo lugar em ambas as strings.

8.8 8. Trecho Final repetido

Dada uma string, retorne a contagem do número de vezes que uma substring de comprimento 2 aparece na string e também como os últimos 2 caracteres da string, então “hixxxhi” produz 1 (não contaremos a substring final).

CHAPTER

NINE

PROBLEMAS PARA O DOJO INICIAL

9.1 1. Ordene uma lista

Crie uma função em Python que aceite dois parâmetros. O primeiro será uma lista de números. O segundo parâmetro será uma string que pode ter um dos seguintes valores: asc, desc e none. Se o segundo parâmetro for "asc", a função deve retornar uma lista com os números em ordem crescente. Se for "desc", a lista deve estar em ordem decrescente e, se for "none", deve retornar a lista original inalterada.

9.2 2. Converta um número decimal em binário

Escreva uma função em Python que aceite um número decimal e retorne o número binário equivalente. Para simplificar, o número decimal sempre será menor que 1.024, portanto, o número binário retornado sempre terá menos de dez dígitos.

9.3 3. Conte as vogais em uma string

Crie uma função em Python que aceite uma única palavra e retorne o número de vogais dessa palavra. Nesta função, apenas a, e, i, o e u serão contados como vogais — não y.

9.4 4. Oculte o número do cartão de crédito

Escreva uma função em Python que aceite um número de cartão de crédito. Ele deve retornar uma string onde todos os caracteres estão ocultos com um asterisco, exceto os quatro últimos. Por exemplo, se a função for enviada "4444444444444444", ela deverá retornar "############4444".

9.5 5. Os Xs são iguais aos Os?

Crie uma função Python que aceite uma string. Esta função deve contar o número de Xs e o número de Os na string. Ele deve retornar um valor booleano de True ou False. Se a contagem de Xs e Os for igual, a função deve retornar True. Se a contagem não for a mesma, deve retornar False. Se não houver Xs ou Os na string, ela também deve retornar True porque 0 é igual a 0. A string pode conter qualquer tipo e número de caracteres.

9.6 6. Crie uma função de calculadora

Escreva uma função Python que aceite três parâmetros. O primeiro parâmetro é um número inteiro. O segundo é um dos seguintes operadores matemáticos: +, -, / ou . O terceiro parâmetro também será um número inteiro. A função deve realizar um cálculo e retornar os resultados. Por exemplo, se a função for passada 6 e 4, ela deve retornar 24.

9.7 7. Dê-me o desconto

Crie uma função em Python que aceite dois parâmetros. O primeiro deve ser o preço total de um item como um número inteiro. O segundo deve ser a porcentagem de desconto como um número inteiro.

A função deve retornar o preço do item após a aplicação do desconto. Por exemplo, se o preço for 100 e o desconto for 20, a função deve retornar 80.

9.8 8. Apenas os números

Escreva uma função em Python que aceite uma lista de qualquer tamanho que contenha uma mistura de inteiros não negativos e strings. A função deve retornar uma lista apenas com os inteiros da lista original na mesma ordem.

9.9 9. Repita os caracteres

Crie uma função Python que aceite uma string. A função deve retornar uma string, com cada caractere na string original duplicado. Se você enviar a função “agora” como parâmetro, ela deve retornar “aaggoorraa”, e se você enviar “123a!”, ela deve retornar “112233aa!!”.

CHAPTER

TEN

PROBLEMAS PARA O DOJO INTERMEDIÁRIO

10.1 1. Números da Sorte

Os números da sorte são um subconjunto de números inteiros. Em vez de entrar em muita teoria, vejamos o processo para chegar aos números da sorte:

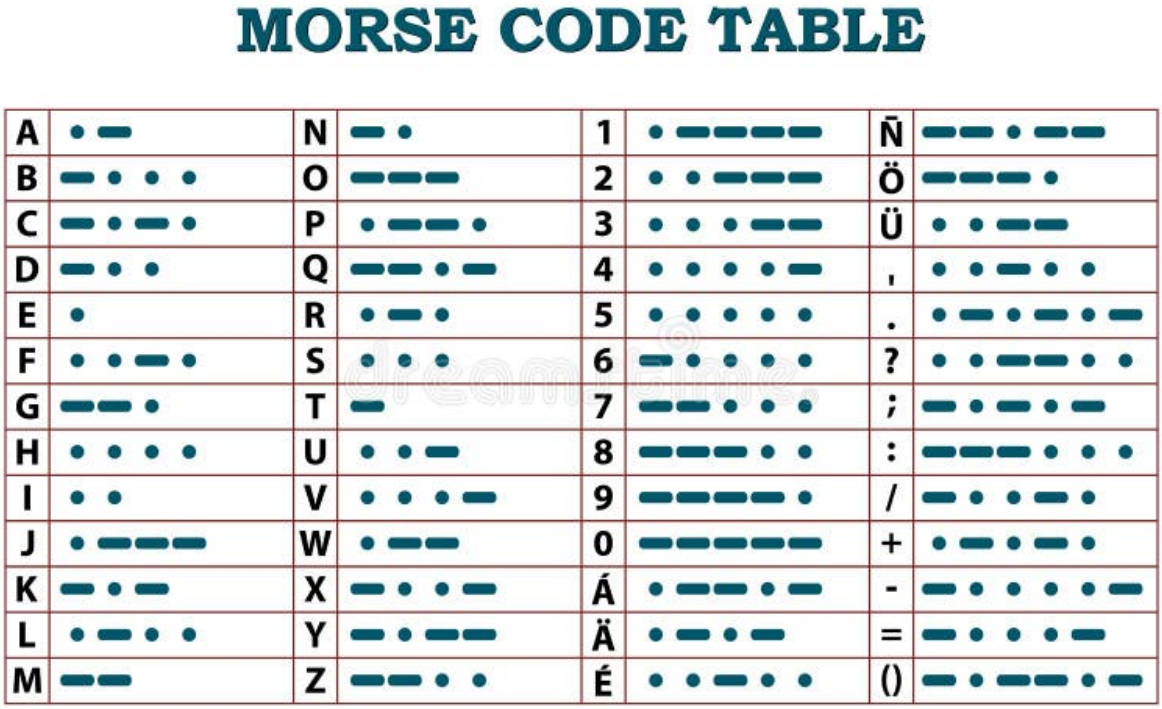
> Pegue o conjunto de inteiros 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,..... Primeiro, exclua cada segundo número, obtemos o seguinte conjunto reduzido. 1,3,5,7,9,11,13,15,17,19,..... Agora, exclua cada terceiro número, obtemos 1, 3, 7, 9, 13, 15, 19,..... Continue este processo indefinidamente..... Qualquer número que NÃO seja excluído devido ao processo acima é chamado de "sortudo". Portanto, o conjunto de números da sorte é 1, 3, 7, 13,.....

Dado um número inteiro n, escreva uma função para dizer se esse número é sortudo ou não.

10.2 2. Converta para código morse

Não usamos mais o código Morse para transferir informações, mas isso não significa que você não possa usá-lo em um desafio de código. Escreva uma função em Python que receba uma string que pode ter caracteres alfanuméricos em letras minúsculas ou maiúsculas.

A string também pode conter quaisquer caracteres especiais tratados em código Morse, incluindo vírgulas, dois pontos, apóstrofos, pontos, pontos de exclamação e pontos de interrogação. A função deve retornar o código Morse equivalente para a string.



10.3 3. Bloqueador Tic Tac Toe

Neste desafio Python, escreva uma função que aceite dois números. Esses números representarão uma posição em um tabuleiro de jogo da velha. Eles podem ser de 0 a 8, onde 0 é o ponto superior esquerdo e 8 é o ponto inferior direito.

Esses parâmetros são duas marcas no tabuleiro do jogo da velha. A função deve retornar o número da vaga que pode impedir que essas duas vagas ganhem o jogo.

10.4 4. Reorganize o número

Para completar este desafio, escreva uma função que aceite um número como parâmetro. A função deve retornar um número que é a diferença entre o maior e o menor número que os dígitos podem formar no número.

Por exemplo, se o parâmetro for “213”, a função deverá retornar “198”, que é o resultado de 123 subtraído de 321.

10.5 5. A moeda falsificada

Você recebeu doze moedas, mas uma é falsificada e o peso dela é ligeiramente diferente das outras. Usando uma balança de dois pratos, identifique qual moeda é a falsa e se seu peso é superior ou inferior ao normal. Desenvolva um programa que descubra a moeda falsa em três pesagens.

10.6 6. Os três mercadores muçulmanos

Disse o xeique, apontando para os três muçulmanos:

- **Aqui estão, ó Calculista, os três amigos. São criadores de carneiros em Damasco.**
  Enfrentam agora um dos problemas mais curiosos que tenho visto. E esse problema é o seguinte: - Como pagamento de pequeno lote de carneiros, receberam aqui, em Bagdá, uma partida de vinho, muito fino, composta de 21 vasos iguais, sendo: 7 cheios; 7 meio cheios; 7 vazios. Querem, agora, dividir os 21 vasos de modo que cada um deles receba o mesmo número de vasos e a mesma porção de vinho. Repartir os vasos é fácil. Cada um dos sócios deve ficar com sete vasos. A dificuldade, a meu ver, está em repartir o vinho sem abrir os vasos, isto é, conservando-os exatamente como estão.

Desenvolva um programa que retorne três listas de vasos, cada lista deve obedecer às regras dadas.

> Malba Tahan, O homem que Calculava

10.7 7. O problema do joalheiro

Um homem que veio da Síria vender jóias em Bagdá prometeu ao dono de uma hospedagem que pagaria 20 dinares pela hospedagem se vendesse as jóias por 100 dinares, pagando 35 se as vendesse por 200 dinares. Mas acabou vendendo tudo por 140 dinares. Quanto deve pagar pela hospedagem então?

Desenvolva um programa que calcule o preço a ser pago

> Malba Tahan, O homem que Calculava

10.8 8. As pilhas de moedas, uma falsa

Suponha-se que temos 10 pilhas de moedas. Uma das pilhas é inteiramente formada de moedas falsas, mas não sabemos qual é essa pilha. Sabemos apenas que as moedas falsas pesam uma grama a menos que as genuínas. A balança é uma que tem um ponteiro e diz quanto é o peso no prato.

Desenvolva um programa que em uma única pesagem diga qual é a pilha de moedas falsas.

> Malba Tahan, O homem que Calculava

10.9 9. O problema dos 8 pães

João deseja pagar Pedro e Marcus que durante uma viagem repartiram com ele 8 pães, sendo que Pedro tinha 5 pães e Marcus tinha 3. A divisão foi feita de modo uniforme e cada pão custa 1 moeda. Quantas moedas Pedro e Marcus devem receber?

Desenvolva um programa que retorne o valor devido a Pedro e a Marcus.

> Malba Tahan, O homem que Calculava

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search