|  |
| --- |
| Projeto piloto |
| Introdução a linguagem de programação Python |
| Em quatro capítulos |
|  |
| **André Ricardo Prazeres Rodrigues** |
| **1/8/2019** |

|  |
| --- |
| XXX |

Metas

Reconhecer a informática como ferramenta para novas estratégias de aprendizagem, capaz de contribuir de forma significativa para o processo de construção do conhecimento nas diversas áreas.

Objetivos

Compreender as funções básicas dos principais produtos de automação da microinformática, tais como sistemas operacionais, interfaces gráficas, bancos de dados, planilhas de cálculos, linguagem HTML e redes de computadores.

Sumário

[Capítulo 1 O Python 4](#_Toc361666747)

[1.1. A história do Python 4](#_Toc361666748)

[1.2. Instalação e configuração 5](#_Toc361666749)

[1.2.1. Instalação 5](#_Toc361666750)

[1.2.2. Utilizando o IDLE (Integraded DeveLopment Enviromente) 5](#_Toc361666751)

[1.3. Entrada/Saída de dados 6](#_Toc361666752)

[1.3.1. Input() 6](#_Toc361666753)

[1.3.2. print() 7](#_Toc361666754)

[1.4. Variáveis 7](#_Toc361666755)

[1.4.1. Operadores lógicos 7](#_Toc361666756)

[1.4.2. Tipagem dinâmica 8](#_Toc361666757)

[1.4.3. Strings 8](#_Toc361666758)

[1.4.4. Listas 9](#_Toc361666759)

[1.4.5. Tuplas 9](#_Toc361666760)

[1.4.6. Dicionários 9](#_Toc361666761)

[Capítulo 2 Estrutura de dados 11](#_Toc361666762)

[2.1. Teste (if...elif...else...) 11](#_Toc361666763)

[2.2. Loop (while...) 12](#_Toc361666764)

[2.3. Varredura (for...) 12](#_Toc361666765)

[2.4. Controle de fluxo (pass, break, continue) 13](#_Toc361666766)

[2.5. Tratamento de erro (try...except...) 14](#_Toc361666767)

[2.6. Funções 16](#_Toc361666768)

[2.6.1. A palavra reservada def 16](#_Toc361666769)

[2.6.2. Funções prontas (math) 16](#_Toc361666770)

[2.6.3. O módulo do sistema (os) 17](#_Toc361666771)

[Capítulo 3 Orientação a objetos 18](#_Toc361666772)

[3.1. Classe 18](#_Toc361666773)

[3.2. Atributos e métodos 18](#_Toc361666774)

[3.3. O método especial \_\_init\_\_ (construtor) 19](#_Toc361666775)

[3.4. Encapsulamento/herança 20](#_Toc361666776)

[3.5. Pilha em Python (stack) 21](#_Toc361666777)

[3.6. Fila em Python (Queue) 22](#_Toc361666778)

[Capítulo 4 Conexão ao banco de dados 24](#_Toc361666779)

[4.1. Objeto CRUD em Banco de Dados com python 24](#_Toc361666780)

[4.2. Acesso ao banco de dados SQL 24](#_Toc361666781)

[4.3. Aplicação do CRUD 25](#_Toc361666782)

1. O Python

O propósito deste capítulo é apresentar a criação do Python...

* 1. A história do Python

Antes de falar da linguagem python, vamos passar um pouco da visão de seu criador Guido Van Rossum.

Rossum trabalhou com a linguagem ABC no Centro de Matemática e Ciência da computação, conhecido como CWI, no final de 1982 em Amisterdan. Este instituto de pesquisa era famoso pela invenção da linguagem Algol 68. Com aproximadamente 5 anos o projeto ABC não deu certo e ele foi para grupo Amoeba, que era um sistema distribuído de microkernel, liderado por Sape Mullender. Em 1991, foi para um grupo de multimídia do CWI liderado por Dick Bulterman.

“Python é um produto direto da minha experiência no CWI. ...ABC me deu a inspiração crucial para Python, Amoeba a motivação imediata e o grupo de multimídia fomentou seu crescimento” (Rossum, 2009).

“Minha motivação para a criação de Python foi a percepção da necessidade de uma linguagem de alto nível no projeto Amoeba. Percebi que o desenvolvimento de utilitários para administração de sistema em C estava tomando muito tempo. E fazê-los no shell Bourne não funcionaria por diversas razões. O mais importante foi que, sendo um sistema distribuído de microkernel, as operações primitivas do Amoeba eram bem diferentes (e refinadas) que as operações primitivas disponíveis no shell Bourne. Portanto, havia necessidade de uma linguagem que "preencheria o vazio entre C e o shell". Por um tempo longo, esse foi o principal lema de Python” (Rossum, 2009).

Então, Rossum influenciado pelas linguagens de programação Algol 60, Pascal, Algol 68 e ABC resolveu projetar uma linguagem que abarcaria todo que gostou, principalmente da ABC e consertar alguns problemas que percebia.

O nome Python surgiu por conta do grupo de comédia favorito “Monty Python’s Flying Circus”. Seguindo uma tradição do grupo que trabalhou (Amoeba no CWI) de nomear linguagens com nomes de programas de TV.

“Por muitos anos eu resisti à tentação de associas a linguagem com cobras. Finalmente desisti quando O'Reilly queria colocar uma cobra na capa de seu primeiro livro de Python "Programming Python". Era tradição da O'Reilly usar imagens de animais e, se deveria ser um animal, que fosse uma cobra” (Rossum, 2009).

Em 20 de fevereiro de 1991, Python foi liberado, sob uma licença que era quase uma cópia exata da licença MIT usada pelo projeto X11 na época, para o mundo pela primeira vez pelos grupos de notícias. Em março de 1993 o grupo comp.lang.python foi criado com meu encorajamento, mas sem meu envolvimento direto. No verão de 1994, o grupo estava agitado com uma discussão entitulada "Se Guido fosse atingido por um ônibus?" sobre a dependência da crescente comunidade de Python em relação às minhas contribuições pessoais. Em novembro de 1994, Rossun fez seu primeiro workshop de Python.

Porque estudar?

Código livre

Alto nível

Tipagem dinâmica

Orientado a objetos

Multiplataforma

Documentação

Mercado de trabalho

* 1. Instalação e configuração

Para obter o Python, entre no site: python.org

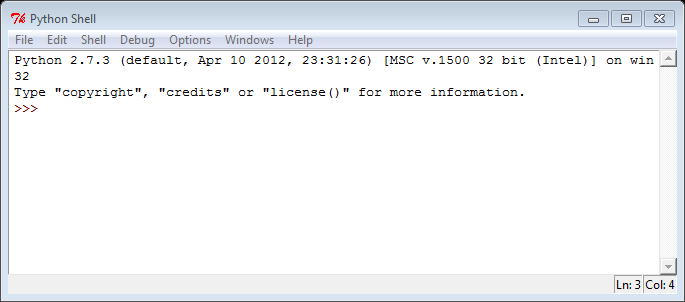
Para o curso adotou-se como base as apostilas “Módulo A – Bem vindo ao Python” e “Módulo B – Python orientado a objetos”. Estas apostilas se encontram no site: python.org.br na seção “Interessante”. Após clicar neste link, vá para a seção documentação Python (formato para impressão). Baixe os módulos A e B.

* + 1. Instalação

O Windows instala o Python por padrão no drive c:\ (raiz) na pasta Python27.

* + 1. Utilizando o IDLE (Integraded DeveLopment Enviromente)

No menu de instalação do Python (versão 2.7) e clique no link “IDLE (Python GUI)”, geralmente está em INICIAR->PROGRAMAS->PYTHON27->IDLE. Abrirá uma janela conforme a figura A.



Falarei hoje da IDE que vem junto com o Python, o IDLE.

IDLE vem da palavra Integrated DeveLopment Environment, e segundo o Wikipedia, é uma alusão ao Eric Idle, um dos membros fundadores.

Esta IDE tem uma interface simples, leve e limpa, justamente para os inciantes na programação, mas os programadores avançados também encontram nele o que precisam em uma IDE.

Os recursos desta IDE vem desde Multi-Janela (podendo abrir vária janelas de programação) com múltiplos "desfazer" (cada janela tem seu histórico), colorização dos comandos Python, e outras várias capacidades como identação, chamadas e autocomplemento

O IDLE executa o código Python em um processo separado, que é reiniciado a cada RUN(F5) iniciado no editor. Garantindo que o reinicio do shell não interfira no editor ou IDLE.

A IDE contem um debugger com passos, breakpoints persistentes e pilhas de chamadas visíveis.

Aqui também podemos configurar a fonte de texto, cores, opções de inicializações e atalhos.

O IDLE é feito 100 % em Python, usando a ferramenta GUI Tkinter (Tk/Tcl) e em multiplataforma, funciona tanto em Unix, Mac e Windows.

1 - Menu

1.1 - File Menu

New Window - Cria uma nova janela de edição.

Open... - Abre um arquivo existente

Open Module... - Abre um modulo existente (busque sys.path)

Class Browser - Mostra as classes e os métodos no arquivo atual

Path Browser - Mostra os diretórios sys.path, módulos, classes e métodos.

Save - Salva o conteúdo da janela atual associada a uma extensão(Quando o conteúdo não está salva, aparece o "\*" ao lado do titulo)

Save As... - Save o conteúdo da janela atual em um novo arquivo, podendo alterar a extensão.

Save Copy As... - Salva o conteúdo da janela atual em um novo arquivo sem alterar a extensão do arquivo.

Close - Fecha a janela atual (questiona se quer salvar quando o conteúdo não estiver salvo).

Exit - Fecha todas as janelas e sai do IDLE (questiona se quer salvar quando o conteúdo não estiver salvo).

1.2 - Edit Menu

Undo - Desfaz a ultima alteração na janela atual.

Redo - Refaz a ultima alteração na janela atual.

Cut - Recorta trechos selecionados para o clipboard.

Copy - Copia trechos selecionados para o clipboard.

Paste - Cola o trecho armazenado no clipboard na janela.

Select All - Seleciona todo o conteúdo da janela. (Similar aoCtrl+A)

Find... - Abre a caixa de pesquisa com várias opções.

Find Again - Repete a ultima busca.

Find Selection - Procura por uma string em uma seleção.

Find in Files... - Abre uma caixa de pesquisa para aquivos.

Replace... - Abre uma caixa de pesquisa e substituição.

Go To Line - Mostra a linha desejada.

Indent Region - Desloca a linha selecionada em 4 espaços para a direita.

Dedent Region - Desloca a linha selecionada em 4 espaços para a esquerda.

Comment out Region - Insere ## no inicio das linhas selecionadas.

Uncomment Region - Remove o # ou ## das linhas selecionadas.

Tabify Region - Espaço entre as abas.

Untabify Region - Coloca todas as abas no numero certo de espaço.

Expand Word - Expande a palavra que você digitou para encontrar outra palavra; repetir para ter uma expansão diferente.

Format Paragraph - Reformo paragrafo na linha em branco selecionada.

Import Module - Importa ou recarrega o módulo atual.

Run Script - Executa um arquivo no \_\_main\_\_.

1.3 - Windows Menu

Zoom Height - Alterna a janela entre o tamanho normal (24x80) e o máximo.

O restante deste menu são os mesmo de todas as janelas.

1.4 - Debug Menu (Na janela Python Shell somente)

Go To File/Line - Verifica um ponto de um arquivo e um numero da linha, abre o arquivo, e a mostra.

Open Stack View - Mostra a pilha da ultima exceção.

Debugger Toggle - Roda os comandos do Shell sob o debugger.

JIT Stack View Toggle - Abre o visualizador de pilha no rastreador.

2 - Edições Básicas e Navegação

\* Backspace deleta o caractere para a da direita; Del da esquerda

\* Page Up/Page Down e as Setas para se movimentar

\* Home/End vai para o inicio/fim da linha

\*C-Home/C-End para inicio/fim do arquivo

\* Alguns atalhos Emacs podem também funcionar, incluindo C-B,C-P,C-A, C-E, C-D, C-D.

2.1 - Indentação Automática

Apos uma declaração de bloco-aberto, o proxima linha terá 4 espaços. Apos, certas palavras (Break,Return e etc.) a linha é retornada 4 espaços.

2.2 - Python Shell Window

C-C - Interrompe a execução do comando

C-D - Envia o arquivo final; fecha a janela se digitado no prompt

Alt-P - Refaz o ultimo comando com as mesmas características

Alt-N - Retenta o próximo.

Return - Enquanto estiver num comando anterior, retenta este comando.

Alt-/ - Expande a palavra, muito usado aqui.

3 - Cores das Sintaxes

A coloração é aplicada em uma "linha"no background, mas pode ocasionar de o texto não obter cor.

Para alterar as cores, edite o trecho "colors" na seção dentro do "config.txt"

Sintaxes Python, Cores:

\* Keywords

Alaranjado

\* Strings

Verde

\* Comentários

Vermelho

\* Definições

Azul

Cores do Shell

\* Saída do Console

Marrom

\* Stdout

Azul

\* Stderr

Verde Escuro

\* Stdin

Preto

4 - Inicio

Quando inicializado com a opção "-s", O IDLE executara o arquivo referenciado pela variável de ambiente IDLESTARTUP ou PYTHONSTARTUP. Idle primeiramente checa pelo "IDLESTARTUP"; se estiver presente o arquivo referenciado rodará. Caso não estiver presente, o idle procura pelo "PYTHONSTARTUP". Arquivos referenciados pelas variáveis ambiente são postos para armazenar as funções que são usadas frequentemente no shell idle, ou para executar uma demonstração ou um modulo muito usado.

Em adição, o modulo "tk"só carrega um arquivo de inicialização quando o encontra. Note que o arquivo Tk é carregado incondicionalmente. Este arquivo adicional é o ".Idle.py" e é visto para os usuários no diretório raiz do usuário.

4.1 - Comandos de uso

idle.py [ -c comando ] [-d] [-e] [-s] [-t titulo] [argumento] ...

-c comando : Executa "comando"

-d : Ativa Debugger

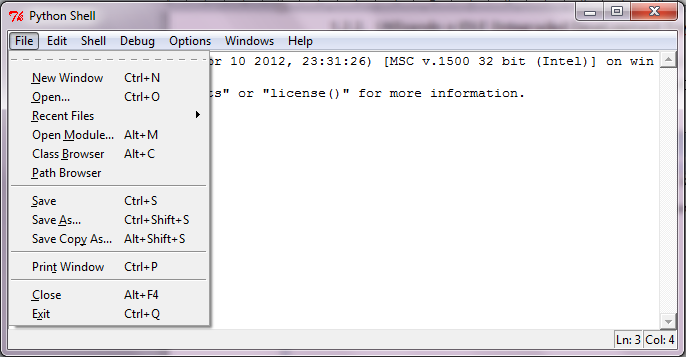
-e : Modo Edit; Argumentos são arquivos para ser editados

-s : Roda $IDLESTARTUP ou $PYTHONSTARTUP primeiros

-t titulo : Define o titulo da janela

Por hoje foi explicado o IDLE. Caso este post contenha algum erro, não deixe de comunicar.

O menu File é o principal item deste IDLE.



* 1. Entrada/Saída de dados
     1. Input()

Comando de entrada de dados.

Sintaxe:

>>> input([prompt])

Retorna sempre uma string.

Exemplo :

>>> x = input(“Entre um valor: ”)

Entre com um valor:

No IDLE ficaria conforme a figura X

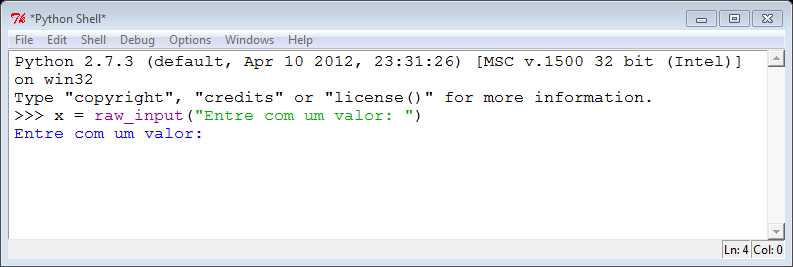


Figura X: Python Shell

Na versão Python 3.x o comando seria:

>>> x = Input(“Entre com um valor: ”)

Entre com um valor:

* + 1. print()

Comando de saída de dados.

Sintaxe:

print([prompt])

Exemplo :

>>> print(“Olá mundo!”)

Olá mundo!

* 1. Estrutura de dados

[definir o conceito de estrutura de controle]

* + 1. Operadores lógicos

>>> a = 1

>>> b = 2

>>> a == b # igualdade

False

>>> a!= b # diferença

True

>>> a > b # maior que

False

>>> a < b # menor que

True

>>> 2\*a > b #

False

>>> 2\*\*3 # exponenciação

8

>>> 2\*\*(3+6)

512

>>> 7 % 2 # resto da divisão

1

* + 1. Tipagem dinâmica

>>> v = 1

>>> type(v)

<type ‘int’>

>>> v = 4.

>>> type(v)

<type ‘float’>

>>> v = “Boa tarde”

>>> type(v)

<type ‘str’>

>>> v = []

>>> type(v)

<type ‘list’>

>>> v = ()

>>> type(v)

<type ‘tuple’>

>>> v = {}

>>> type(v)

<type ‘dict’>

* + 1. Strings

Pag. 7-9

capitalize() # inicia a primeira letra da palavra como maiúscula.

isdigit() # retorna verdadeiro ou falso se a variável é um número.

split() # divide o texto em palavras no formato de lista.

upper() # transforma o texto em caixa alta.

replace() # troca o caráter desejado por outro.

count() # conta quantos caracteres tem dentro da variável string

startswith() # verifica a inicial da variável string.

Formatadores de saída de string (interpolação) com o operador “%”.

Sintaxe:

print(“texto %s” % (variável))

Exemplo :

>>> v = 50

>>> n = “André”

>>> print(“Você ganhou %d de %s” % (v,n))

Símbolos de interpolação

%s – texto

%d – inteiro

%f – real

%o – octal

%x – hexadecimal

%e – real exponencial

%% – sinal de porcentagem

* + 1. Listas

São elementos entre colchetes separados por vírgula.

Operações com listas

lst = [] # lista vazia

lst[i] = 3 # substitui um elemento

lst[i:j] = 5 #substitui um grupo de elementos

del [i:j] # remove um grupo de elementos

lst.append(“a”) # adiciona um elemento

lst.extend([1,2,3]) # adiciona uma lista

lst.index(3) # retorna o número índice do elemento

lst.insert(i,x) # insere um elemento na posição x

lst.pop() # retire o último elemento

lst.remove(x) # um elemento

lst.reserve() # organiza a lista em ordem decrescente

lst.sort() # organiza a lista em ordem crescente

* + 1. Tuplas
    2. Dicionários

São um conjunto de objetos (chave: valor), separados por vírgula entre chaves {}.

Exemplo :

{‘andré’: 39,}

Operações com dicionários

a = {‘nome’:’André’, ‘idade’: 39}

len(a) # número de elementos

a[‘nome’] # valor de nome

a[‘nome’] = ‘Pedro’ # atribui um valor (Pedro) para chave “nome”

del a[‘nome’] # deleta a chave e seu valor

a.clear() # apaga todos os elementos

a.copy() # retorna uma cópia do dicionário

a.items() # retorna uma lista de tuplas (chave,valor) de todos os elementos

a.keys() # retorna uma lista de todas as chaves

a.values() # retorna uma lista de todos os valores

1. Estrutura de controle
   1. Teste (if...elif...else...)

É uma estrutura que executa determinados blocos de comandos na dependência de sua condição (expressão) ser verdadeira.

Sintaxe:

If (expressão):

Bloco comando

[elif (expressão):

Bloco de comando

Else:

Bloco de comando]

Exemplo :

>>> a = 1

>>> if a== 1:

print(“É o número um”)

else:

print(“não é o número um”)

Exemplo :

>>> a = input(“Entre com um número”)

>>> if int(a) % 2 == 0:

print(“Número par”)

else:

print(“número ímpar”)

Vamos agora passar a usar o editor de texto do IDLE.

Exemplo :

usuário = [“andré”,”joão”,”maria”]

username = input

if username in usuário:

print(“acesso permitido”)

else:

print(“acesso negado”)

Exemplo :

a = input(“Que horas são? ”)

if a < 10:

print(“Bom dia!”)

elif a < 12:

print(“Hora de almoçar!”)

elif a < 18:

print(“Boa tarde!”)

elif a < 22:

print(“Boa noite!”)

else:

print(“Vá dormir!”)

Exercícios:

* 1. Loop (while...)

É uma estrutura de repetição que executa um bloco de comando enquanto a condição (expressão) for verdadeira.

Sintaxe:

while (expressão):

Bloco de comando

[else:

Bloco de comando]

Exemplo :

a = 0

while a < 10:

print(a)

a = a + 1 # a += 1

Exemplo :

b = 1

while b < 5:

print(“%i dólares valem %.2f reais” % (b, b\*2.50))

Exercícios

* 1. Varredura (for...)

É uma estrutura de varredura de objetos sequenciáveis (range(), string, lista, tupla, dicionário etc.).

Sintaxe:

for variável in sequência:

Bloco de comando

[else:

Bloco de comando]

Exemplo1:

for i in range(3):

print(i)

Exemplo2:

a = “Curso de Python”

for i in a:

print(i)

Exemplo3:

a = [3, 10, 4]

for i in a:

print(i)

Exemplo4:

a = (“João”, “Pedro”, “Lucas”)

for i in a:

print(i)

Exemplo5:

orcamento = {“luz”:150, “água”:60, “telefone”:108}

for z in orcamento:

print(“%s custa R$ %.2f” % (z, orcamento[z]))

Exemplo6:

matriz = ((1, 0, 0), (0, 1, 0), (0, 0, 1))

for i in range(len(matriz)):

print(“\n”)

for j in range(len(matriz)):

print(matriz[i][j]),

* 1. Controle de fluxo (pass, break, continue)

Pass – define um bloco de comando vazio.

Sintaxe:

z = 0

if z == 1:

pass

break – termina o loop imediatamente, usado somente nas estruturas while e for.

Sintaxe:

a = 0

while a < 10:

a += 1

if a == 5:

break

print(a)

Continue – volta ao início do bloco de comando e continua a próxima interação, usado somente nas estruturas while e for.

a = 0

b = 0

while a < 10:

a += 1

if b >= 5:

Continue

b += 1

print(a, b)

Exercícios:

E1 – Construa um programa que receba uma palavra. Se esta palavra tiver até 5 caracteres apresente somente as vogais, caso contrário apresente as consoantes.

E2 – d

* 1. Tratamento de erro (try...except...)

É uma estrutura usada para capturar exceções.

Sintaxe:

Try:

Bloco de comandos

Except:

Bloco de comandos

Exemplo 11:

lst = []

for i in range(3):

lst.append(int(input(“Digite um valor: ”)))

Exemplo 12:

# coding=cp1252

lst = []

for i in range(3):

while True:

try:

lst.append(int(input(“Digite um valor: ”)))

break

except:

print(“Digite um número!!!”)

Exemplo :

# coding=1252

produto = [“pão”, “leite”, “carne”]

soma = contador = 0

while contador < len(produto):

try:

c = float(input(“Entre com o valor do(a) %s: ” % produto[contador]))

soma = soma + c

contador = contador + 1

except:

print(“Entre somente com um valor!”)

print(“O valor de sua compra foi %s” % soma)

Lista compreensiva

Nome = [“Pedro”,”João”,”Lucas”]

For i in nome:

Print(i)

Juntar = []

Valores = [1, 3, 4, 8]

For i in valores:

Juntar.append(i\*3)

Exemplo:

[i\*3 for i in valores]

* 1. Funções

O que é função?

* + 1. A palavra reservada def

Sintaxe:

def nomeFunção([argumentos]):

Bloco de comando

Exemplo:

>>> def iNome(n):

print(n)

return n # retorna o valor de n

>>> nome = iNome(“André”)

André

>>> nome

André

A variável “nome” receberá o retorno da função iNome.

Exemplo:

import math

def hipotenusa(a, b):

h = math.sqrt(a\*\*2 + b\*\*2)

return h

Exemplo:

n = [[2, 2], [2,3], [3,2]]

for a, b in n:

print(a, b)

hipotenusa(a, b)

Lista compreensiva

[hipotenusa(a, b) for a, b in n]

* + 1. Funções prontas (math)

O módulo matemático (math) apresenta um conjunto de funções:

math.sqrt(n) – retorna a raiz quadrada.

math.cos(n) – retorna o cosseno do número em radiano.

math.sin(n) – retorna o seno do número em radiano.

math.tan(n) – retorna a tangente do número em radiano.

math.radians(n) – converte ângulo(grau) para radiano.

math.hypot(x, y) – retorna a hipotenusa.

math.pi – retorna o PI(3,1416...).

Exercício

1 – Calcule e exiba a área do círculo (A = pi\*r2) de qualquer raio que for digitado.

2 – Calcule o volume do cilindro de raio 6 cm e altura 5 cm (V = pi\*r2\*h).

* + 1. O módulo do sistema (os)

1. Orientação a objetos
   1. Classe

Sintaxe:

Class NomeClasse(object):

Pass

Exemplo:

>>> class Cachorro(object):

pass

Para instanciar uma classe basta iniciá-la em uma variável.

Exemplo:

>>> c = Cachorro()

Cada instância tem um número que o identifica, para saber digite o comando:

>>> id(c) # número inteiro (hex) que identifica o objeto

* 1. Atributos e métodos

Os atributos são objetos inerentes à classe, ou seja, as variáveis.

Exemplo:

class Cachorro(object):

alimento = “carne”

habitat = “doméstico”

nome = “Rex”

>>> from cachorro import Cachorro

>>> c = Cachorro()

>>> c.alimento

carne

>>> c.habitat

doméstico

>>> c.nome

Rex

Os métodos são funções que interagem com os atributos dentro da classe.

Exemplo:

class Circulo(object):

raio = 25,4

def calculaArea(self):

self.area = 3,14\*(self.raio\*\*2)

def calculaVolume(self, altura):

self.volume = 3,14\*(self.raio\*\*2)\*altura

O self não é uma palavra reservada, é um primeiro argumento, é uma maneira de referenciar a própria instância.

Exemplo:

>>> C1 = Circulo()

>>> C1.raio

25,4

>>> C1.area # erro

>>> C1.calculaArea()

>>> C1.area

* 1. O método especial \_\_init\_\_ (construtor)

Utilizado para definir atributos e métodos no momento em que a classe for criada

Sintaxe:

class NomeClasse(objects):

atributos

def \_\_init\_\_(self, [argumentos]):

Bloco de comandos

Exemplo:

class Cachorro(objects):

alimento = “carne”

habitat = “doméstico”

def \_\_init\_\_(self, nome):

self.nome = nome

Exemplo:

>>> c = Cachorro(“Bob”) # ou c = Cachorro(nome=”Bob”)

>>> c.alimento

carne

>>> c.habitat

domestico

>>> c.nome

Bob

Exercício:

Alterar a classe Cachorro para Animais de modo que possamos no momento da instância da classe definir o tipo de animal e suas características

Documentando o arquivo python.

class Teste(objects):

””” Este é um teste de apresentação do atributo especial \_\_doc\_\_.”””

def complemento(self):

””” Pode-se apresentá-lo com o comando help também!”””

print(“help(classe)”)

Exemplo:

>>> help(Teste)

>>> print(Teste.\_\_doc\_\_)

>>> print(Teste().\_\_doc\_\_)

>>> print(Teste().complemento.\_\_doc\_\_)

* 1. Encapsulamento/herança

class A(objects):

a = 1 #atributo público

\_\_b = 2 #atributo privado

class B(A):

\_\_c = 3 # atributo privado a B

def \_\_init\_\_(self):

print(self.a)

print(self.\_\_c)

def \_\_nome(self):

print(self.\_\_c)

def pNome(self):

return self.\_\_nome()

Exemplo:

>>> a = A()

>>> print(a.a)

1

>>> b = B()

1

3

>>> print(b.\_\_b) # erro, \_\_b é privado de A

>>> print(b.\_\_c) # erro, \_\_c é privado de B só chamado pela classe

>>> print(b.\_\_nome()) # erro, método privado de B

>>> print(b.pNome())

* 1. Pilha em Python (stack)

Estrutura chamada LIFO (Last In – First Out)

Operações básicas:

\* TOP – acessa o elemento posicionado no topo da pilha;

\* PUSH – insere um elemento no topo da lista;

\* POP – remove o elemento do topo da lista.

Elementos

FIGURA

Arquivo pilha.py que contém a classe Stack.

class Stack(objetct):

””” Método de inicialização que cria uma lista vazia quando instanciada. ”””

def \_\_init\_\_(self):

self.\_\_pilha = []

””” Método que insere um objeto ao final da pilha.”””

def pushObj(self, obj):

self.\_\_pilha.append(obj)

print(self.\_\_pilha)

””” Método que retorna e remove o último elemento da pilha. ”””

def popObj(self):

if (self.\_\_pilha != []):

return self.\_\_pilha.pop()

else:

print(“Pilha vazia!!!”)

””” Método que mostra a pilha. ”””

def showObj(self):

print(self.\_\_pilha)

Exercício:

>>> from pilha import Stack

>>> p = Stack()

>>> p.pushObj(3)

>>> p.pushObj(8)

>>> p.pushObj([1, 7])

>>> p.pushObj((“nome”, “telefone”))

>>> p.pushObj(“Aula”)

>>> p.popObj()

>>> p.popObj()

>>> p.showObj()

* 1. Fila em Python (Queue)

Estrutura chamada FIFO (First In – First Out)

Operações básicas:

\* Enqueue – insere um elemento no final da fila.

\* Dequeue – remove um elemento do começo da fila

Elementos

FIGURA

Arquivo fila.py que contém a classe Queue.

class Queue(object):

””” Método de inicialização que cria uma fila vazia quando instanciada. ”””

def \_\_init\_\_(self):

self.\_\_fila = []

””” Método que insere um objeto ao final da fila. ”””

def enqueueObj(self, obj):

self.\_\_fila.append(obj)

print(self.\_\_fila)

””” Método que retorna e remove o primeiro elemento da fila. ”””

def dequeueObj(self):

if (self.\_\_fila != []):

return self.\_\_fila.pop(0)

else:

print(“Fila vazia!!!”)

””” Método que mostra a fila.”””

def showObj(self):

print(self.\_\_fila)

Exercício:

>>> from fila import Queue

>>> f = Queue()

>>> f.enqueueObj(3)

>>> f.enqueueObj (8)

>>> f.enqueueObj ([1, 7])

>>> f.enqueueObj ((“nome”, “telefone”))

>>> f.enqueueObj (“Aula”)

>>> f.dequeueObj()

>>> f.dequeueObj()

>>> f.showObj()

1. Conexão ao banco de dados
   1. Objeto CRUD em Banco de Dados com python

Instruções SQL:

Create – INSERT INTO

INSERT INTO tabela (campo1[, ...]) VALUES (valor1[, ...])

Read – SELECT

SELECT \* FROM tabela WHERE (campo1 = valor1)

Update – UPDATE

UPDATE tabela SET campo = valorNovo WHERE (campo1 = valor1)

Delete – DELETE

DELETE FROM tabela WHERE (campo1 = valor1)

* 1. Acesso ao banco de dados SQL

Primeiro deve-se importar os drivers(módulos) do banco que irá usar:

MySQL – import MySQLdb

Oracle – import cx\_Oracle

PostgreSQL – import psycopg2

SQLite3 – import sqlite3

As conexões.

MySQL

con = MySQLdb.connect(“servidor”, “usuário”, “senha”, “banco”)

Oracle

con = cx\_Oracle.connect(“usuário/senha@servidor”)

PostgreSQL

con = pyscopg2.connect(host=”servidor”, user=”usuário”, password=“senha”, dbname=”banco”)

SQLITE 3

con = sqlite3.connect(“banco”)

Obter uma transação (cursor)

cur = con.cursor()

Executar um SQL (execute)

cur.execute(“ALGUM SQL”)

Salvar a transação (commit)

con.commit()

Capturar resultados (fetch)

res = cur.fetchone() # retorna um linha

res = cur.fetchall() # retorna todas as linhas

res = cur.dictfetchall # retorna as linha no formato de um dicionário.

* 1. Aplicação do CRUD

# coding=cp1252

import sqlite3

# Criar uma conexão e uma transação

conexão = sqlite3.connect(“aula.sqlite”)

cursor = conexao.cursor()

# Criar uma tabela, antes verificar sua existência.

cursor.execute(“select name from sqlite\_master”)

resultado = cur.fetchall()

for i in resultado:

if (i == “animais”):

print(“Tabela animais já existe”)

else:

sql = ”””create table animais

(id interger primary key,

Nome varchar(100),

Habitat varchar(50))”””

cursor.execute(sql)

# Sentença SQL para inserir dados via SQLite

sql = “insert into animais values (null, ?, ?)”

# Valores que serão registrados na tabela

valores = [(“Soneca”, “Doméstico”),(“Kate”, “Selvagem”)]

# Inserir valores

for rec in valores:

cursor.execute(sql, rec)

# Confirma a transação

conexao.commit()

# Selecionar registros

cursor.execute(“select \* from animais”)

resultados = cursor.fetchall()

For rec in resultados:

Print(“%d: %s(%s)” % rec)

# Alterar registros

# Via Sqlite

sql = “update animais set nome=?, habitat=? Where id=?”

valores = (“Berola”, “Selva”, 1)

cursor.execute(sql, valores)

conexao.commit()

# Via python

valores = (“Toto”, “Híbrido”, 2)

sql = “update animais se nome=’%s’, habitate=’%s’ where id=’%i’” % (valores[0], valores[1], valores[2])

cursor.execute(sql)

conexao.commit()

# Deletar registro

sql = “delete from animais where id=?”

valores = (2,)

cursor.execute(sql, valores)

conexao.commit()

# Limpar as operações pendentes no banco

conexao.rollback()

# Fechar a conexão

conexao.close()

Agora vamos criar uma classe CRUD, genérica, capaz de receber uma conexão e poder executar as quatro operações básicas em uma tabela.

operacao.py

# coding= cp1252

"""O objetivo desta classe é fazer a operação CRUD (criar, ler, alterar e deletar)."""

class Operacao:

"""Método construtor que recebe a conexão do banco de dados"""

def \_\_init\_\_(self, conexao):

self.conexao = conexao

self.cursor = conexao.cursor()

"""Método destinado a inserir registros na tabela especificada conforme os campos escolhidos"""

def inserir(self, tabela, campos, dados):

"""Gerar o SQL para inclusão"""

valores = ''

for i in campos:

valores+=', ?'

sql = "INSERT INTO %s VALUES (null%s)" % (tabela, valores)

print(sql)

"""Executa SQL, insere registros"""

self.cursor.execute(sql, dados)

"""Confirma a transação"""

self.conexao.commit()

"""Método destinado a consultar registros da tabela especificada conforme os campos escolhidos"""

def consultar(self, tabela, campos='', dados=''):

"""

Verifica-se se foi passado os parâmetros (campos e dados).

O 'if' testa se não foi passado os parâmetros (campos e dados), caso positivo, faz uma pesquisa geral da tabela;

O 'elif' testa se foi passado somente um parâmetro (campos e dados), faz uma pesquisa especifica;

O 'else' pressupõe-se mais de um parâmetro (campos e dados), faz uma pesquisa por mais de um campo.

Gerar o SQL para pesquisa.

"""

if campos == '':

sql = "SELECT \* FROM %s" % (tabela)

elif len(campos) == 1:

sql = "SELECT \* FROM %s WHERE %s = '%s'" % (tabela, campos[0], dados[0])

else:

pesquisa = ''

for i in range(len(campos)):

pesquisa+="%s = '%s' AND " % (campos[i], dados[i])

sql = "SELECT \* FROM %s WHERE %s" % (tabela, pesquisa[:-5])

print(sql)

"""Executa a SQL, Seleciona registros"""

self.cursor.execute(sql)

"""Recupera os resultados"""

resultados = self.cursor.fetchall()

"""Mostra"""

for rec in resultados:

print (rec)

return resultados

"""Método destinado a alterar registros da tabela especificada conforme os campos escolhidos"""

def alterar(self, tabela, campos, dados):

"""Gerar o SQL para pesquisa e alteração"""

pesquisa = ''

for i in range(1,len(campos)):

pesquisa+="%s = '%s', " % (campos[i], dados[i])

sql = "UPDATE %s SET %s WHERE %s=%i" % (tabela, pesquisa[:-2], campos[0], dados[0])

print(sql)

"""Executa a SQL, altera registros"""

self.cursor.execute(sql)

"""Confirma a transação"""

self.conexao.commit()

"""Método destinado a anterar registros da tabela especificada conforme os campos escolhidos."""

def deletar(self, tabela, campos, dados):

"""Gerar o SQL para deleção"""

sql = "DELETE FROM %s WHERE %s = %s" % (tabela, campos[0], dados[0])

print(sql)

"""Executar a SQL, deleta registros"""

self.cursor.execute(sql)

"""Confirma a transação"""

self.conexao.commit()

Exercício:

import sqlite3

conexao = sqlite3.connect(“aula.sqlite”)

from operação import Operacao

o = Operacao(conexao)

campos = (“nome”, “habitat”)

valores = (“Totó”, “Lama”)

o.inserir(“animais”, campos, valores)

o.consultar(“animais”)

campos = (“id”, “nome”)

valores = (1, “Rex”)

o.alterar(“animais”, campos, valores)

campos = (“id”,)

valores = (1,)

o.deletar(“animais”, campos, valores)