LINGUAGEM DE PROGRAMAÇÃO II
CCET115
Abstração e Revisão

- Universidade Federal do Acre
- Disciplina: Linguagem de Programação II
- Prof.: Manoel Limeira de Lima Júnior
- Email: juniorlimeiras@gmail.com

Abstração: base crítica da programação

- A programação orientada a objeto baseia-se na identificação e no trabalho com abstrações
- Uma abstração permite que um conceito seja expresso e usado repetidamente sem que todos os detalhes tenham de ser considerados
- **Python** dá suporte ao uso de abstrações através do mecanismo de **classes** e **objetos**
  - **Classes definem** as propriedades e os comportamentos de uma abstração
  - Os **objetos são instâncias** das classes que permitem a utilização da abstração nos programas

Conceito e exemplo de Abstração

- **Abstração** é um processo mental pelo qual atemos aos aspectos mais importantes de alguma coisa, ao mesmo tempo em que ignoramos os menos importantes

Uma floresta.

Uma árvore.

Uma folha.

Abstração em Python

- Existem abstrações básicas, sem motivo para recriá-las, Python possui uma gama de funções e bibliotecas de classes com diversos objetivos que economizam tempo e esforços

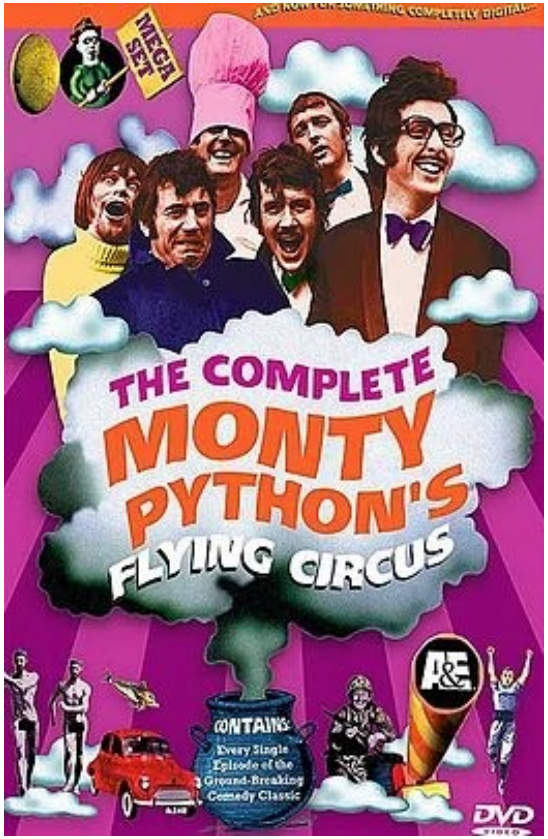
```
len()
```

- Encontrar e construir abstrações é a chave para o desenvolvimento de sistemas bem sucedidos, deve-se portanto aprender e usar as bibliotecas de classe apropriadas

```
def minha_funcao():
    pass
```

Python

- É uma das linguagens de programação mais usadas
- Permite programar nos paradigmas Orientado a Objetos e Procedimental
- Os programas geralmente são menores que outras linguagens de programação
- Está sendo usada por quase todas as empresas gigantes da tecnologia como – Google, Amazon, Facebook e Instagram
- Fornece uma coleção de bibliotecas que podem ser usadas para diversos fins:
  - Aprendizado de máquina e mineração de dados (como Pandas, TensorFlow, Scikit-learn)
  - Aplicativos GUI (como Tkinter, PyQt, PyGUI, PyGtk)
  - Estruturas da Web como Django (usado pelo YouTube, Instagram, Dropbox)
  - Processamento de imagem (como OpenCV, Pillow)
  - Raspagem da Web (como Scrapy, BeautifulSoup, Selenium)
  - Jogos (como PyGame, PySoy)



Porque aprender Python?

- Linguagem de alto nível multiparadigma com excelente legibilidade, simples, clara e poderosa
- Versátil, fácil de ler, aprender e escrever
- Estruturas de dados amigáveis
- Tipada dinamicamente
- Orientada a objetos
- Interpretada
- Multiplataforma
- Altamente eficiente

Recursos Gratuitos

- *Python.org*
- *Codecademy*
- *Learnpython.org*
- *FreeCodeCamp*
- *Google's Python Class*
- *Tutorials Point*
- *W3Schools*
- *Pythonspot*
- *Full Stack Python*

Variáveis em Python

- Área de memória que mantém um valor e pode ser mudado.
- **Identificador da variável**: sequência de caracteres, iniciada por uma letra minúscula
  - Exemplos: nota, saldo, deposito, saque
- **Padronização**: nomes de variáveis devem ser minúsculos, não deve ter acento e separados por *underlines*. Esse padrão é chamado de ***snake case***

```
altura = 1.75
primeiro_nome = "Limeira"
x = 1000


print(altura)
print(primeiro_nome)
print(x)
```

Tipos Básicos (Embutidos)

- Tipos básicos são imutáveis
- Inteiro (**`int`**): limitado pela memória do computador
- Booleano (**`bool`**): 0 é **False** e qualquer outro valor é **`True`**
- Ponto-Flutuante (**`float`**): números reais
- String (**`str`**): sequência de caracteres iniciada e terminada por aspas simples ou duplas
- O **None** é um valor do tipo NoneType, é usado para representar a abstenção de um valor.
- Experimente a função type()
  - `type('')`

Conversão de Tipos

- String, booleano ou ponto-flutuante para inteiro
  - **`int()`**
    - `int(3.14)`
- String, booleano ou inteiro para ponto-flutuante
  - **`float()`**
    - `float("123.45")`
- String, inteiro ou ponto-flutuante para booleano
  - **`bool()`**
    - `bool(0)`
- Booleano, inteiro ou ponto-flutuante para String
  - **`str()`**
    - `str(123.45)`

Strings

- Strings multi linha: aspas triplas habilita o string multi-linha em seu código:
  - `"""Este é um comentário com várias linhas:`
    `A função`
    `é um bloco de código`
    `que visa organizar as ideias"""`
- O operador **+** é utilizado tanto para concatenar strings quando os dois operandos são do tipo **str**. No entanto, quando um operador é string e o outro for um número, o interpretador Python indica um erro.
- Sequências de escape para strings:
  - `\n`: nova linha
  - `\t`: tabulação
  - `\'` e `\"`: aspas simples e aspas duplas.
  - `\\`: barra

Operadores

- Soma (**+**)
- Subtração (**-**)

operadores aditivos

- Produto (*****)
- Divisão de ponto flutuante (***/***)
- Divisão inteira (***//***)
- Resto da divisão inteira (**%**)
- Potenciação (******)

operadores multiplicativos

- Atribuição aumentada:

```
a += 10  # mesmo que a = a + 10
```

  Vale para todos os operadores: `-=, *=, /=, //=, %= e **=`

- Operadores Lógicos:
  - Disjunção ou soma lógica (**or**)
  - Conjunção ou produto lógico (**and**)

Operadores Binários

- Igual a (**==**)
- Diferente de (**!=**)
- Maior que (**>**)
- Menor que (**<**)
- Maior ou igual a (**>=**)
- Menor ou igual a (**<=**)

O resultado de uma operação relacional é um valor booleano.

```
(2 + 2) == 5 #False
(2 + 2) <= 5 #True
'abd'>'abc'  #True
```

- a **is** b
  - True se a e b são idênticos
- a **is not** b
  - True se a e b não são idênticos
- a **in** b
  - True se a é membro de b
- a **not in** b
  - True se a não é membro de b
- Exemplos:

```
x = [1, 2, 3]
y = [1, 2, 3]
x == y #True
x is y #False
x in y #False
```

Precedência dos Operadores

Maior Prioridade

1. Expressões entre Parênteses **( )** #da esquerda para a direita
2. Potenciação (**) #da direita para a esquerda
3. Unários (**+, -**) #da direita para a esquerda
4. Binários Multiplicativos (***, /, //, %**) #da esquerda para a direita
5. Binários Aditivos (**+, -**) #da esquerda para a direita
6. Relacionais (**==, !=, <, >, <=, >=**)
7. Lógico **not**
8. Lógico **and**
9. Lógico **or**

Menor Prioridade.

`5.75 + a%b - 7/8.1` **equivale a** `5.75 + (a%b) - (7/8.1)`

Comandos de Entrada e Saída

- **`input()`**
  - Comando aguarda o usuário fornecer, pela entrada padrão (teclado), um valor expresso por uma sequência de caracteres, e o retorna
- **`input(`*`mensagem`*`)`**
  - `idade = input(“Digite sua idade: ”)`
- **`print()`**
  - Pula para a próxima linha na saída padrão (vídeo)
- **`print(`*`expressão`*`)`**
  - Escreve na saída padrão (vídeo) e pula para a próxima linha
- **`print(`*`exp1, exp2, ..., expN`*`)`**
  - Escreve na saída padrão o resultado da avaliação de cada expressão *expNum*; Um espaço em branco é escrito entre cada par de *expNum*; Ao final, pula para a próxima linha

Comandos de Saída Padrão

- **print(***expressão*, **end =** *término***)**
  - Escreve na saída padrão (vídeo) o resultado da avaliação da *expressão*
  - Ao final, escreve a String de *término*

- **print(***exp1*, *exp2*, ..., *expN*, **end =** *término***)**
  - Escreve na saída padrão (vídeo) o resultado da avaliação de cada expressão *expNum*
  - Um espaço em branco é escrito entre cada par de *expNum*
  - Ao final, escreve a String de *término*

- **print(***exp1*, *exp2*, ..., *expN*, sep **=** *separador***)**
  - Escreve na saída padrão (vídeo) o resultado da avaliação de cada expressão *expNum*
  - Um separador é escrito entre cada par de *expNum*

Expressões Formatadas (operador %(...))

- Exemplos:
  - **print(**“A média dos números é %.2f” **% (**82432.923421**))**
    - Escreve “A média dos números é 82432.92” e pula de linha
  - **print(**“%f + %f = %4.1f” **% (**15,7.8313, 15+7.8313**))**
    - Escreve “15.000000 + 7.831300 = 22.8” e pula de linha
  - **print(**“%d + %d = %d” **% (**5.89,7.83,5.89+7.83**), end=**“!”**)**
    - Escreve “5 + 7 = 13!” (apenas a parte inteira) e não pula de linha
- Outra maneira mais elegante (versão 3) é usar a função **format()**:
  - **print(**'A média dos números é {:.2f}'.format**(**82432.923421**))**
  - **print(**'{f} + {f} = {:4.1f}'.format**(**15, 7.8313, 15+7.8313**))**
  - **print(**'{d} + {d} = {d}'.format**(**5.89,7.83,5.89+7.83**), end=**“!”**)**

Expressões Formatadas (operador %(...))

- A partir da versão 3.6 Python tem uma terceira forma de compor strings, chamada: **f-string**. Escrevemos a letra **f** antes de abrirmos as aspas e o nome da variável diretamente na string entre **{}**
- As regras de formatação são as mesmas da função **format**

```
nome = ‘Manoel’

idade = 32

salario = 1200.25

print(f’{nome} tem {idade} anos e R$ {salario:4.2f} no bolso’)
```

Estruturas em Python

- Python tem 4 tipos de Estruturas de Dados embutidas: **Lista, Tupla, Dicionário e Conjunto**
- **Lista (list)** é uma estrutura de dados **mutável**, ou seja, os itens podem ser adicionados após a sua criação. É como se você fosse fazer compras no mercado e fizesse uma lista de itens e mais tarde você pudesse adicionar mais itens à lista
- A função **`append()`** é usada para adicionar dados à lista

```
lista = []

lista = list()

lista.append("Carne")

lista.append("Pão")

lista.append("Cerveja")

print(lista)
```

Estruturas em Python

- Outras funções úteis

`lista.insert(i, x)` Insere um item em uma dada posição i.

`lista.remove(x)` Remove o primeiro elemento, cujo valor seja x.

`lista.pop(i)` Remove o item de posição i da lista e o retorna. Caso i não seja especificado, retorna o último elemento da lista.

`lista.clear()` Remove todos os elementos da lista.

`lista.count(x)` Retorna o número de vezes que o valor x aparece na lista.

`lista.sort(reverse=False):` Ordena os elementos da lista.

`lista.reverse()` Reverte os elementos da lista.

`lista.copy():` Retorna uma lista com a cópia dos elementos da lista de origem.

Estruturas em Python

- **Tuplas (tuple)** são estruturas heterogêneas e **imutáveis,** definidas de forma parecida com uma lista com a diferença do delimitador. Enquanto listas utilizam colchetes, as **tuplas usam parênteses** (opcionais)

```
tupla1 = (1, 2)                 #declaração
tupla2 = 'três', 4, 'cinco'     #empacotamento
len(tupla2)                     #tamanho
x, y = tupla1                   #desempacotamento
tupla3 = tupla1 + tupla2        #concatenação
4 in tupla3                     #pertencimento
lista1 = list(tupla3)           #transformação
tupla4 = tuple(lista1)
```

Estruturas em Python

- **Dicionários (dict)** são estruturas não ordenadas que associam um conjunto de pares: uma chave a um valor
- As chaves são **únicas, imutáveis e podem ser strings, números e tuplas**, mas não listas

```
dic = {}
dic = dict()
dic.update({"Carne":5})
dic.update({"Pão":10})
dic["Cerveja"] = 48
print(dic)
```

- Funções importantes: `keys()`, `values()`, `items()`, `copy()`, `clear()`, `get(chave)`, `pop(chave)`
- Para ordenar um dicionário pelas chaves pode-se usar a função `sorted()` passando o nome do dicionário como parâmetro

Estruturas em Python

- **Conjuntos (set)** são coleções **não ordenadas e que não admitem elementos duplicados**
- Os conjuntos são **mutáveis** o que permite alterar, excluir ou adicionar itens, porém, seus itens devem ser imutáveis – como strings, inteiros ou tuplas

```
c1 = set()          #declaração
c2 = {1, 3, 5, 7} #declaração
c1.add(2)           #inserção
c1.add(4)
c2.remove(7)        #remoção
7 in c2             #pertencimento
c1 - c2             #diferença
c1 | c2             #união
c1 & c2             #intersecção
c1 ^ c2             #diferença simétrica
```

List Comprehension

- Consiste em um **par de colchetes** contendo uma expressão seguida de uma **cláusula for, e então zero ou mais cláusulas for ou if**. O resultado será uma nova lista resultante da avaliação da expressão no contexto das cláusulas for e if
- Fornece uma sintaxe mais curta para criar uma nova lista com base nos valores de uma lista existente
- Vantagens
  - Mais eficiente em termos de tempo e espaço que loops
  - Requer menos linhas de código
  - Transforma a instrução iterativa em uma fórmula
- Sintaxe

```
lista_nova = [ expressão(item) for item in lista if condição ]
```

List Comprehension

- Exemplo com if

```
lista = [i**i for i in range(11) if i % 2 == 0]
print(lista)


#transformando em laço
lista = []
for i in range(11):
    if i % 2 == 0:
        lista.append(i**i)
```

List Comprehension

- Exemplo com if e else

```
lista = [i if i%2==0 else i*i for i in range(11)]
print(lista)

#transformando em laço
lista = []
for i in range(11):
    if i % 2 == 0:
        lista.append(i)
    else:
        lista.append(i*i)
```

List Comprehension

- Outros exemplos

```
lista = [-4, -2, 0, 2, 4]
[x*2 for x in lista]            #Nova lista com valores dobrados
[-8, -4, 0, 4, 8]
[x for x in lista if x >= 0]    #Lista com valores não negativos
[0, 2, 4]
[abs(x) for x in lista]         #Aplicar uma função para os elementos
[4, 2, 0, 2, 4]

frutas = ['  banana', '  abacate ', 'abacaxi  ']
[item.strip() for item in frutas]
['banana', 'abacate', 'abacaxi']

[(x, x**2) for x in range(6)]   #Lista de tuplas (elemento, quadrado)
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]
```

List Comprehension

- Exemplo com matrizes

```
matriz = [[i+1+j*3 for i in range(3)] for j in range(3)]
print([x for linha in matriz for x in linha])
print(matriz)    #[[1,2,3], [4,5,6], [7,8,9]]

matriz = [[i+1+j*4 for i in range(4)] for j in range(3)]
print([linha[x] for linha in matriz for x in range(4)])
print(matriz) #[[1,2,3,4], [5,6,7,8], [9,10,11,12]]
```

List Comprehension

- Exemplo com matrizes

```
transposta = []
for i in range(4):
    coluna = []
    for linha in matriz:
        coluna.append(linha[i])
    transposta.append(coluna)
transposta
#ou simplesmente
[[linha[i] for linha in matriz] for i in range(4)]
```

List Comprehension

```
import time
def loop(n):                #Função usando loop
    result = []
    for i in range(n):
        if i % 2 == 0:
            result.append(i*i)
    return result
def listc(n):               #Função usando list-comprehension
    return [i*i for i in range(n) if i % 2 == 0]
inicio = time.time()    #captura do tempo de inicio
loop(10**7)
fim = time.time()       #captura do tempo final
print('Tempo com loop:', round(fim – inicio, 2))
inicio = time.time()
listc(10**7)
fim = time.time()
print('Tempo com list-comprehension', round(fim – inicio, 2))
```

List Comprehension

- Exemplo com Conjunto

```
frase = 'Exemplo de list comprehension com set'
vogais_todas = [c for c in frase if c in 'aeiou']
vogais_unicas = {c for c in frase if c in 'aeiou'}
vogais_todas
vogais_unicas
```

List Comprehension

- Exemplo com Dicionário

```
quadrados = {}
for i in range(10):
    quadrados[i] = i ** 2
```

```
quadrados = {i: i**2 for i in range(10)}
```

LINGUAGEM DE PROGRAMAÇÃO II
CCET115
Abstração e Revisão

- Universidade Federal do Acre
- Disciplina: Linguagem de Programação II
- Prof.: Manoel Limeira de Lima Júnior
- Email: juniorlimeiras@gmail.com