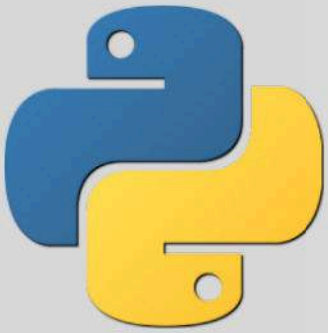
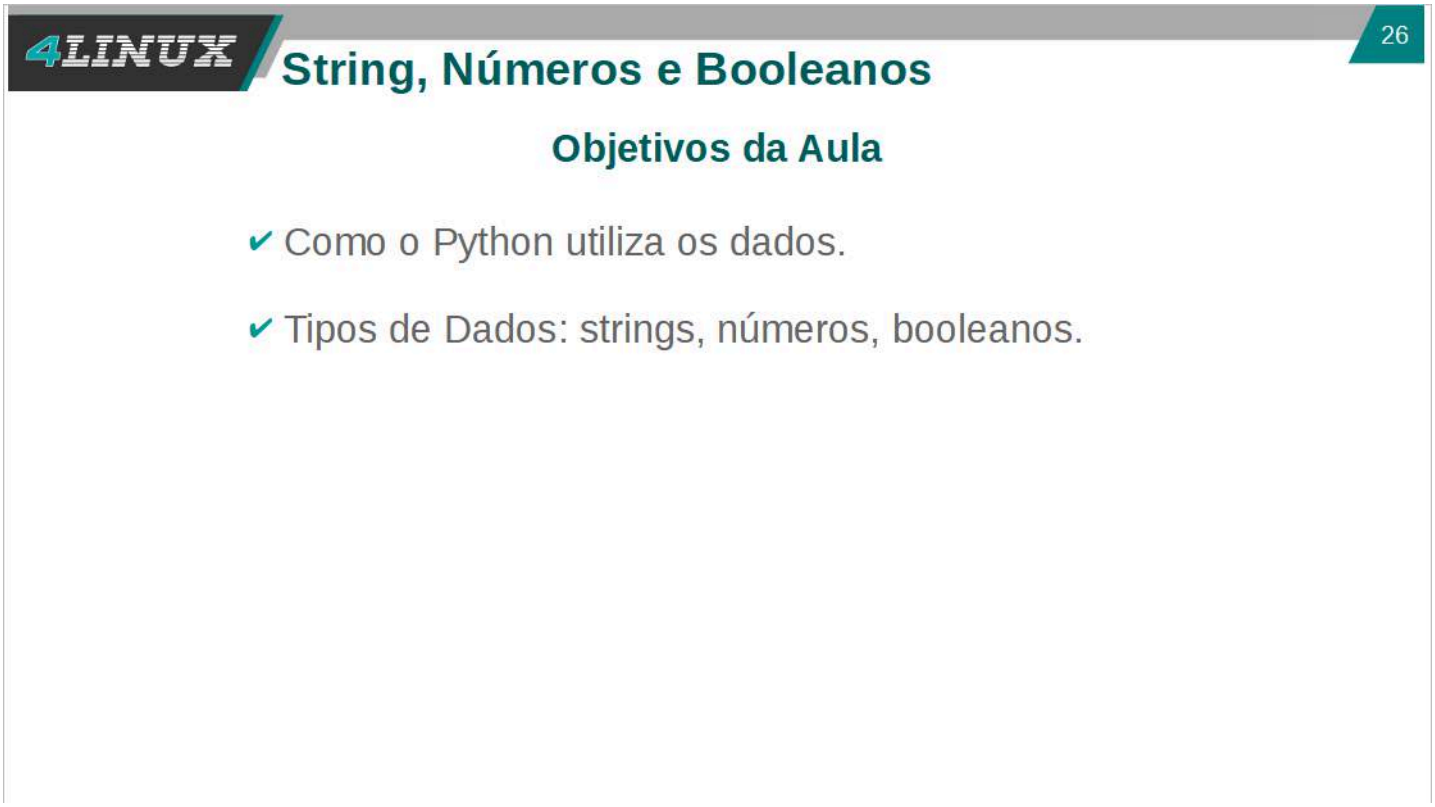
Python Fundamentals



String, Números e Booleanos

2.1

Anotações



Tipos de Dados Básicos

Nesta aula apresentaremos como o Python utiliza os tipos de dados, strings, números e booleanos.

Anotações

Tipos de Dados Básicos

- ✔ O Tipo de dado constitui a maneira como classificamos a informação. Por exemplo: dentro do meu código, números inteiros serão processados diferente de números decimais.
- ✔ Toda a informação é obrigatoriamente de um “tipo”.
- ✔ Em Python, temos os **built-ins**, tipos de dados, embutidos no núcleo da linguagem.

Tipagem de Dados

Muitas linguagens de programação, exigem que as aplicações contenham especificações acerca de qual tipo, constituem determinadas variáveis. Por exemplo: em JAVA para definir um valor inteiro, será necessário utilizar: **“int var = 1;”** outras linguagens que utilizam são: C, C++ e Haskell.

Como vimos na Aula 01, em Python a tipagem de dados é dinâmica, ou seja, não precisamos especificar qual é o tipo de uma informação.

Anotações

Números

Os tipos numéricos no Python são: números inteiros, números de ponto flutuante e números complexos.

```
Inteiros
>>> x = 10
>>> type(x)
<class'int'>
```

```
Flutuantes
>>> x = 3.12
>>> type(x)
<class'float'>
```

```
Complexos
>>> x= 1 - 2j
>>> type(x)
<class'complex'
```

Tipos de Dados - Numéricos

Na linguagem Python, há três tipos de dados numéricos: números inteiros, números de ponto flutuante e números complexos.

Números inteiros, são identificados por **'int',** são descritos como números de precisão fixa.
Números de ponto flutuante, são reconhecidos como **'float',** são números de precisão variável.
Números complexos, diferenciam-se por **'complex',** possuem uma parte real e uma parte imaginária.

Anotações

Booleanos

- ✔ Booleanos em Python são: True (Verdadeiro) e False (Falso).
- ✔ Apesar dessa representação em caracteres, os booleanos False podem representar 0 e True o inteiro 1.
- ✔ Em controle de decisão, Python também considera 0 como False e tudo que for diferente como True.

Anotações

Booleanos

```
In [1]: vdd = True

In [2]: fal = False

In [3]: vdd.numerator
Out[3]: 1

In [4]: fal.numerator
Out[4]: 0

In [5]: not vdd
Out[5]: False

In [6]: not fal
Out[6]: True
```

Tipos de Dados - Booleanos

In[1]: Definimos uma variável que recebeu um booleano verdadeiro.
In[2]: Definimos uma variável que recebeu um booleano falso.
In[3]: Utilizamos o atributo 'numerator' do objeto booleano para ver seu valor numérico.
In[4]: Utilizamos o atributo 'numerator' do objeto booleano para ver seu valor numérico.
In[5]: Utilizamos o operador de negação e objetivamos a saída inversa.
In[6]: Utilizamos o operador de negação e objetivamos a saída inversa.

Anotações

Texto

Dado do tipo texto em Python, é identificado por “string”.

```
String
>>> x = "Python Fundamentals - 4linux"
>>> print(x)
Python Fundamentals - 4linux
>>> type(x)
<class 'str'>
```

Tipos de Dados – Texto

Os dados de texto em Python, são identificados por **‘str’,** podem ser definidos de duas formas:

Citações simples: 'Python Fundamentals – 4linux'
Aspas: “Python Fundamentals – 4linux”

Anotações

Método de String

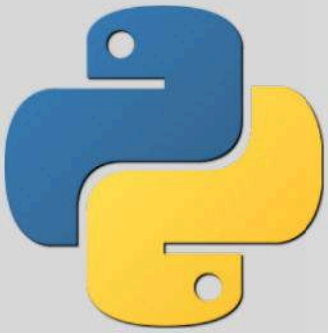
```
In [1]: var = ' curso python fundamentos '

In [2]: var.title()
Out[2]: ' Curso Python Fundamentos '

In [3]: var.upper()
Out[3]: ' CURSO PYTHON FUNDAMENTOS '

In [4]: var.lower()
Out[4]: ' curso python fundamentos '

In [5]: var.replace('o', '0')
Out[5]: ' curs0 pyth0n fundament0s '

In [6]: var.strip()
Out[6]: 'curso python fundamentos'

In [7]: var.split()
Out[7]: ['curso', 'python', 'fundamentos']

In [8]: var.islower()
Out[8]: True

In [9]: var.startswith(' c')
Out[9]: True

In [10]: '.'.join(var)
Out[10]: ' .c.u.r.s.o. .p.y.t.h.o.n. .f.u.n.d.a.m.e.n.t.o.s. '
```

Tipos de Dados – Texto

In [1] : Definimos a variável var que recebe a string.
In [2] : O método ‘title’ deixa a string no formato título.
In [3] : O método ‘upper’ deixa a string no formato uppercase.
In [4] : O método ‘lower’ deixa a string no formato lowercase.
In [5] : O método ‘replace’ recebe dois parâmetros, primeiro o ‘string’ que deseja trocar, segundo ‘string’’ a qual deseja substituir a anterior.
In [6] : O método ‘strip’ elimina espaços vazios antes e dois da ‘string’, temos também o ‘rstrip, elimina espaços vazios na direita e, ‘lstrip’ elimina espaços vazios na esquerda.
In [7] : o método ‘split’, quebra a ‘string’ em uma lista, podendo ser utilizado passando um parâmetro para definir por qual carácter, separar cada elemento da lista, por default este parâmetro recebe um espaço vazio.
In [8] : O método ‘isxxx’ verifica o tipo da ‘string’ e retorna um booleano se verdadeiro ou falso.
In [9] : O método ‘startswitch’ verifica os caracteres iniciais da ‘string’ e devolve um booleano.
In [10] : O método ‘join’ interage uma ‘string’ com a outra , mesclando seus caracteres.

Anotações

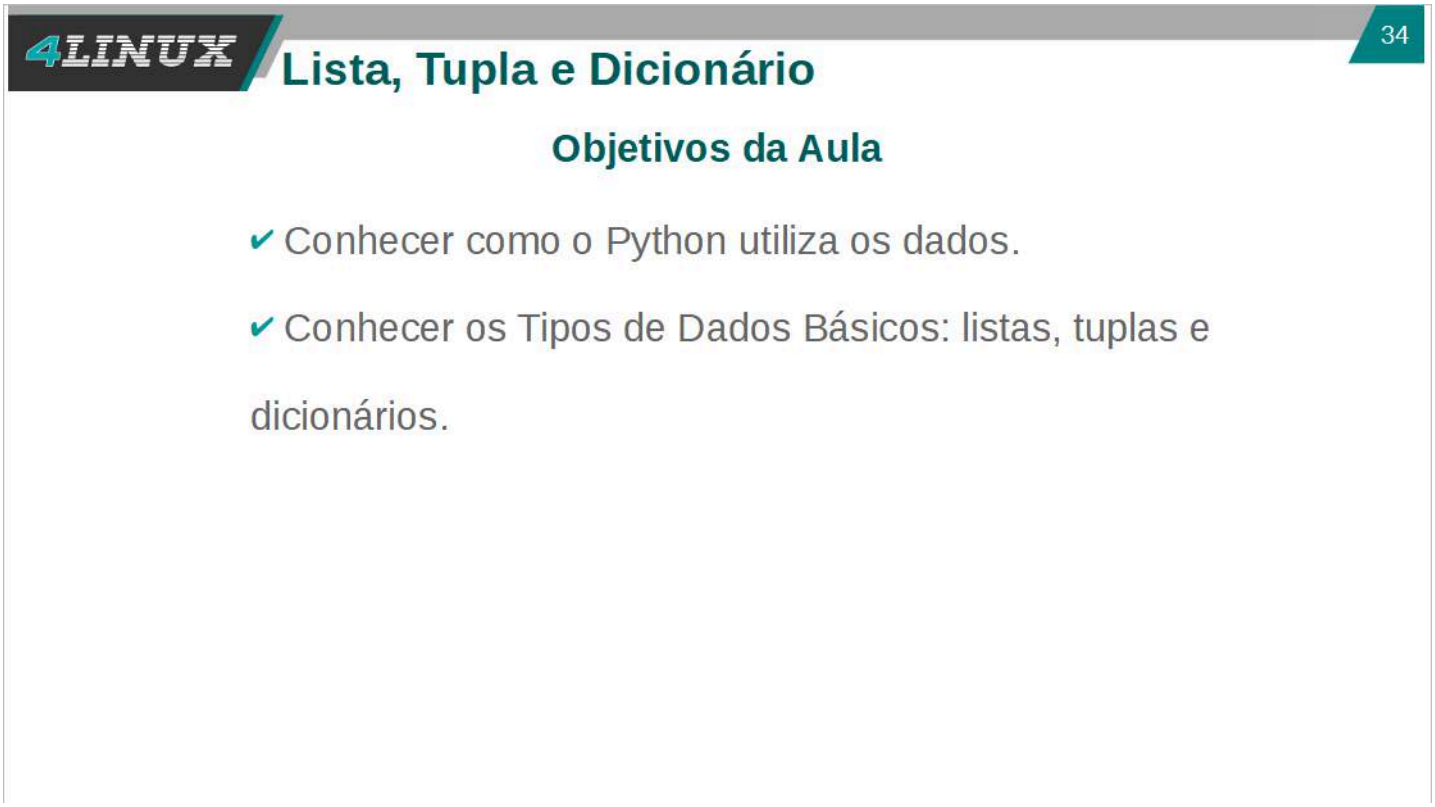
Python Fundamentals



Lista, Tupla e Dicionário

2.2

Anotações



Tipos de Dados Básicos

Nesta aula compreenderemos como o Python utiliza dados em listas, tuplas e dicionários.

Anotações

Lista

Estrutura de dados composta por itens organizados de forma linear, podem ser acessados a partir de um índice que representa sua posição na coleção (iniciando em zero).

```
lista = ['a', 'b', 'c', 'd']
print(lista)
['a', 'b', 'c', 'd']

print(lista[0])
'a'
```

```
print(type(lista))
<class 'list'>

print(type(lista[0]))
<class 'str'>
```

Anotações

Lista

Índice e fatiamento

```
In [1]: lista = ['a', 'b', 'c', 'd']

In [2]: lista[2]
Out[2]: 'c'

In [3]: lista[-1]
Out[3]: 'd'

In [4]: lista[0] = 'A'

In [5]: lista
Out[5]: ['A', 'b', 'c', 'd']

In [6]: lista[:]
Out[6]: ['A', 'b', 'c', 'd']

In [7]: lista[1:]
Out[7]: ['b', 'c', 'd']

In [8]: lista[1:-2]
Out[8]: ['b']
```

Tipos de dados: Índice e fatiamento

In[1]: Definimos uma lista de strings em uma variável.
In[2]: Acessamos um elemento pelo índice.
In[3]: Podemos acessar os índices de trás pra frente, passando números negativos.
In[4]: Podemos alterar o valor do elemento de uma lista atribuindo com sinal de recebe.
In[5]: Visualizamos a lista com alteração realizada no passo anterior.
In[6]: Podemos clonar uma lista, ou uma parte dela, passando o fatiamento, na sintaxe acima o que vem antes do sinal ':' é início, posteriormente é o fim. Caso não especifique, ocorrerá uma cópia do começo ao fim.
In[7]: Especifica o índice de início do fatiamento, ocultando o fim, observe que foi feio um recorte na lista.
In[8]: Fatia a lista especificando início e fim, o início é inclusivo e o fim é exclusivo.

Anotações

Métodos de Lista

```
In [1]: letras = ['a', 'b', 'c', 'd']

In [2]: letras.append('e')

In [3]: letras
Out[3]: ['a', 'b', 'c', 'd', 'e']

In [4]: letras.insert(0,'A')

In [5]: letras
Out[5]: ['A', 'a', 'b', 'c', 'd', 'e']

In [6]: letras.pop()
Out[6]: 'e'

In [7]: letras
Out[7]: ['A', 'a', 'b', 'c', 'd']

In [8]: letras.pop(2)
Out[8]: 'b'

In [9]: letras
Out[9]: ['A', 'a', 'c', 'd']
```

Tipos de dados: Métodos e lista

In[1]: Definimos uma lista de strings em uma variável.
In[2]:Utilizamos o método ‘append’, recebe parâmetro do que será inserido incluindo no último índice da lista.
In[3]: Visualizamos a lista com a alteração realizada no passo anterior.
In[4]: Utilizamos o método ‘insert’ para receber dois parâmetros, primeiro, a posição que deverá ser inserida. Segundo, o que será inserido na posição informada.
In[5]: Visualizamos a lista com a alteração realizada no passo anterior.
In[6]: Utilizamos o método ‘pop’, remove o último índice da lista.
In[7]: Visualizamos a lista com a alteração realizada no passo anterior.
In[8]: Utilizamos o método ‘pop’ especificando a posição do item que deve ser removido.
In[9]: Visualizamos a lista com a alteração realizada no passo anterior.

Anotações

Métodos de Lista

Tipos de dados: Métodos e lista

In[1]: Definimos uma lista de strings em uma variável.
In[2]: Utilizamos o método 'sort' que ordena a lista.
In[3]: Visualizamos a lista com a alteração realizada no passo anterior.
In[4]: Utilizamos o método 'reverse' que inverte a ordem da lista.
In[5]: Visualizamos a lista com a alteração realizada no passo anterior.
In[6]: Utilizamos o método 'index' que recebe um parâmetro de busca e retorna a posição da primeira ocorrência do elemento procurado.
In[7]: Utilizamos o método 'count' que recebe um parâmetro de busca e retorna o número de elementos encontrado.
In[8]: Utilizamos o método 'remove' que recebe um parâmetro de busca e remove a primeira ocorrência encontrada.
In[9]: Visualizamos a lista com a alteração realizada no passo anterior.

Anotações

Tupla

- ✔ Como uma lista, trata-se de uma sequência de itens de qualquer tipo.
- ✔ Entretanto, tuplas são imutáveis, isso as diferencia de listas. Sintaticamente, uma tupla copnsiste uma sequência de valores separados por vírgula.
- ✔ Apesar de não ser necessário, há o entendimento de envolver uma tupla entre parêntese.

Anotações

Tupla

```
In [1]: linguagens = ('python', 'java', 'golang')

In [2]: linguagens
Out[2]: ('python', 'java', 'golang')

In [3]: linguagens[0]
Out[3]: 'python'

In [4]: linguagens[0].title()
Out[4]: 'Python'

In [5]: linguagens[1] = 'javascript'
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-5-a0572093153a> in <module>()
----> 1 linguagens[1] = 'javascript'

TypeError: 'tuple' object does not support item assignment
```

Tipos de dados: Tuplas

In[1]: Atribuímos uma tupla de strings em uma variável.
In[2]: Listamos seus valores.
In[3]: Acessamos seus valores por índice.
In[4]: Utilizamos um método de string, no valor acessado da tupla.
In[5]: Tentamos alterar um elemento da tupla, recebemos uma mensagem de erro, pois as tuplas são imutáveis.

Anotações

Dicionários

- ✔ Constituem um coleção desordenada de objetos.
- ✔ Representados na forma de chave. Esta, é usada para referenciar um determinado valor.
- ✔ As chaves de um dicionário são de tipo imutável como: inteiros, floats e strings.
- ✔ Não possuem uma noção de índice, não podem ser fatiados.
- ✔ São mutáveis, a qualquer momento é possível inserir ou remover itens.

Anotações

Métodos de Dicionário

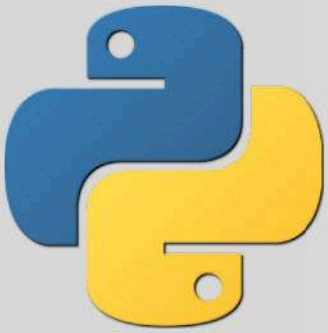
```
In [1]: ling_favorita = {'joao':'java', 'daniel':'python', 'hector':'php'}

In [2]: ling_favorita['daniel']
Out[2]: 'python'

In [3]: ling_favorita['hector']= 'python'

In [4]: ling_favorita
Out[4]: {'joao': 'java', 'daniel': 'python', 'hector': 'python'}

In [5]: ling_favorita.get('joao')
Out[5]: 'java'

In [6]: ling_favorita.keys()
Out[6]: dict_keys(['joao', 'daniel', 'hector'])

In [7]: ling_favorita.values()
Out[7]: dict_values(['java', 'python', 'python'])

In [8]: ling_favorita.items()
Out[8]: dict_items([('joao', 'java'), ('daniel', 'python'), ('hector', 'python')])
```

Tipos de dados: Dicionários

In[1]: Atribuímos um dicionário em uma variável
In[2]: Acessamos um valor específico, utilizando a chave.
In[3]: Atribuímos um novo valor, especificando a chave que receberá a alteração.
In[4]: Listamos o dicionário para visualizar alterações efetuadas no passo anterior.
In[5]: Utilizamos o método 'get' para efetuar uma busca no dicionário.
In[6]: Utilizamos o método 'keys' para listar as chaves do dicionário.
In[7]: Utilizamos o método 'values' para listar os valores do dicionário.
In[8]: Utilizamos o método 'items' para listar chaves e valores do dicionário.

Anotações

Python Fundamentals



Estrutura de Dados e Conversão

2.3

Anotações

Estrutura de dados e Conversão de tipos

Nesta aula apresentaremos a Estrutura dos dados, seus métodos e a conversão de tipos.

Anotações

Estrutura de Dados em Python

- ✔ Em Python podemos definir dois tipos de estruturas: sequências e dicionários.
- ✔ Sequências são dados ordenados, finitos e acessados através de um índice.
- ✔ Dicionários são objetos mapeados através de chaves.

Anotações

Formas de Concatenar Variáveis

Podemos concatenar strings utilizando o método “format”.

```
In [1]: nome, idade = 'Guido', 62

In [2]: mensagem = ' O nome do criador do Python é {0} e sua idade é {1}'.format(nome, idade)

In [3]: mensagem
Out[3]: ' O nome do criador do Python é Guido e sua idade é 62'
```

Concatenando Strings

In [1]: Atribuímos dois valores em duas variáveis, observe que podemos atribuir inúmeras variáveis de uma vez em uma só linha. Neste caso, nome = ‘guido’ e idade = 62
In [2]: Utilizamos o método ‘format’ para concatenar string, dentro da string utilize a sintaxe de {} que serão substituídas por variáveis, que serão passadas dentro do método, neste exemplo, passamos parâmetros posicionais, nas próximas aulas aprenderemos a passar parâmetros nomeados.
In [3]: Visualizamos a variável para confirmar as alterações do passo anterior.

Anotações

Formas de Concatenar Variáveis

Outra forma para concatenar uma string, utiliza o operador **(+)**, porém, em python não é possível concatenar dados de tipos diferentes com este operador.

```
In [1]: nome, idade = 'Guido', 62

In [2]: msg1 = 'O nome do criador do Python é ' + nome + ' e sua idade é ' + str(idade)

In [3]: msg1
Out[3]: 'O nome do criador do Python é Guido e sua idade é 62'
```

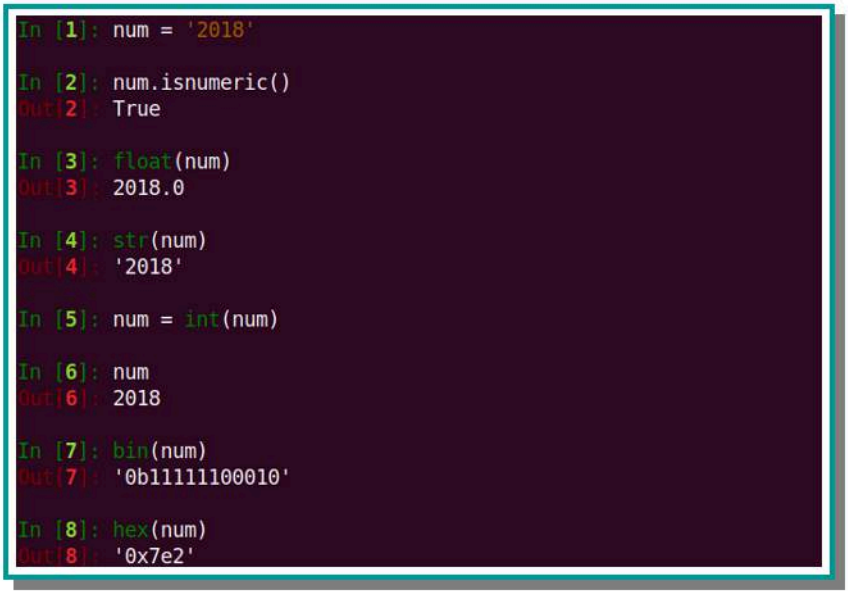
Concatenando Strings

In [1]: Atribuímos dois valores em variáveis.
In [2]: Utilizamos o operador '+', para concatenar as strings, porém, a variável idade do tipo inteiro requer a conversão para 'str' de maneira a evitar erro na concatenação, atribuímos o resultado final em uma variável.
In [3]: Exibimos a variável para visualizar as alterações do passo anterior.

Anotações

Estrutura de dados e Conversão de tipos

Conversão de tipos



Conversões de tipos

In [1]: Atribuímos uma string em uma variável.
In [2]: Utilizamos o método de string 'isnumeric', para verificar se a string é um valor numérico com possibilidade de conversão.
In [3]: Convertemos a string para float.
In [4]: Embora a variável seja string, podemos converter também números e outros tipos para string, utilizando a função str.
In [5]: Atribuímos uma conversão permanente para inteiro.
In [6]: Verificamos a alteração do passo anterior.
In [7]: Convertemos o número para binário
In [8]: Convertemos o número para hexadecimal.

Anotações

Estrutura de dados e Conversão de tipos

Conversão de tipos

```
In [1]: letras = ['a', 'b', 'c', 'd']

In [2]: tuple(letras)
Out[2]: ('a', 'b', 'c', 'd')

In [3]: ling_favorita = {'joao':'javascrip', 'hector':'php', 'daniel':'python'}

In [4]: list(ling_favorita.keys())
Out[4]: ['joao', 'hector', 'daniel']

In [5]: list(ling_favorita.values())
Out[5]: ['javascrip', 'php', 'python']

In [6]: ling_favorita = list(ling_favorita.items())

In [7]: ling_favorita
Out[7]: [('joao', 'javascrip'), ('hector', 'php'), ('daniel', 'python')]

In [8]: type(ling_favorita[0])
Out[8]: tuple

In [9]: dict(ling_favorita)
Out[9]: {'joao': 'javascrip', 'hector': 'php', 'daniel': 'python'}
```

Conversões de tipos

In [1]: Atribuímos uma lista de strings em uma variável.
In [2]: Convertemos a lista para o formato tupla.
In [3]: Atribuímos um dicionário em uma variável.
In [4]: Convertemos as chaves do dicionário em lista.
In [5]: Convertemos os valores do dicionário em lista.
In [6]: Convertemos chaves e valores do dicionário, em lista e atribuímos em uma variável.
In [7]: Verificamos a variável para visualizar as alterações do passo anterior.
In [8]: Verificamos o tipo de dado do índice, 0 da lista, que retorna uma tupla. Logo, temos uma lista de tuplas.
In [9]: Convertemos nossa lista de tuplas para dicionário.

Anotações