Algoritmos e Estruturas de Dados I

Tipos Abstratos de Dados

Prof. Tiago Eugenio de Melo
tmelo@uea.edu.br

www.tiagodemelo.info

Observações

Observações

- O conteúdo dessa aula é parcialmente proveniente do Capítulo 1 do livro “Data Structures and Algorithms using Python”.

Observações

- O conteúdo dessa aula é parcialmente proveniente do Capítulo 1 do livro “Data Structures and Algorithms using Python”.
- As palavras com a fonte `Courier` indicam uma palavra-reservada da linguagem de programação.

Introdução

Introdução

- As linguagens de programação oferecem tipos de dados como uma parte integrante da linguagem.

Introdução

- As linguagens de programação oferecem tipos de dados como uma parte integrante da linguagem.
  - Estes tipos de dados são conhecidos como **tipo primitivo de dados**.

Introdução

- As linguagens de programação oferecem tipos de dados como uma parte integrante da linguagem.
  - Estes tipos de dados são conhecidos como **tipo primitivo de dados**.
  - Esses tipos podem ser: simples ou complexos.

Introdução

- As linguagens de programação oferecem tipos de dados como uma parte integrante da linguagem.
  - Estes tipos de dados são conhecidos como **tipo primitivo de dados**.
  - Esses tipos podem ser: simples ou complexos.
  - Inteiros e reais são tipos simples.

Introdução

- As linguagens de programação oferecem tipos de dados como uma parte integrante da linguagem.
  - Estes tipos de dados são conhecidos como **tipo primitivo de dados**.
  - Esses tipos podem ser: simples ou complexos.
  - Inteiros e reais são tipos simples.
  - Os tipos complexos são construídos a partir de múltiplos tipos primitivos ou mesmo com outros tipos de dados complexos.

Introdução

- As linguagens de programação oferecem tipos de dados como uma parte integrante da linguagem.
  - Estes tipos de dados são conhecidos como **tipo primitivo de dados**.
  - Esses tipos podem ser: simples ou complexos.
  - Inteiros e reais são tipos simples.
  - Os tipos complexos são construídos a partir de múltiplos tipos primitivos ou mesmo com outros tipos de dados complexos.
  - Em Python, objetos, strings, listas e dicionários são exemplos de tipos complexos.

Introdução (cont.)

Introdução (cont.)

- Muitas vezes, os tipos primitivos oferecidos pelas linguagens de programação não são suficientes para resolver problemas maiores ou mais complexos.

Introdução (cont.)

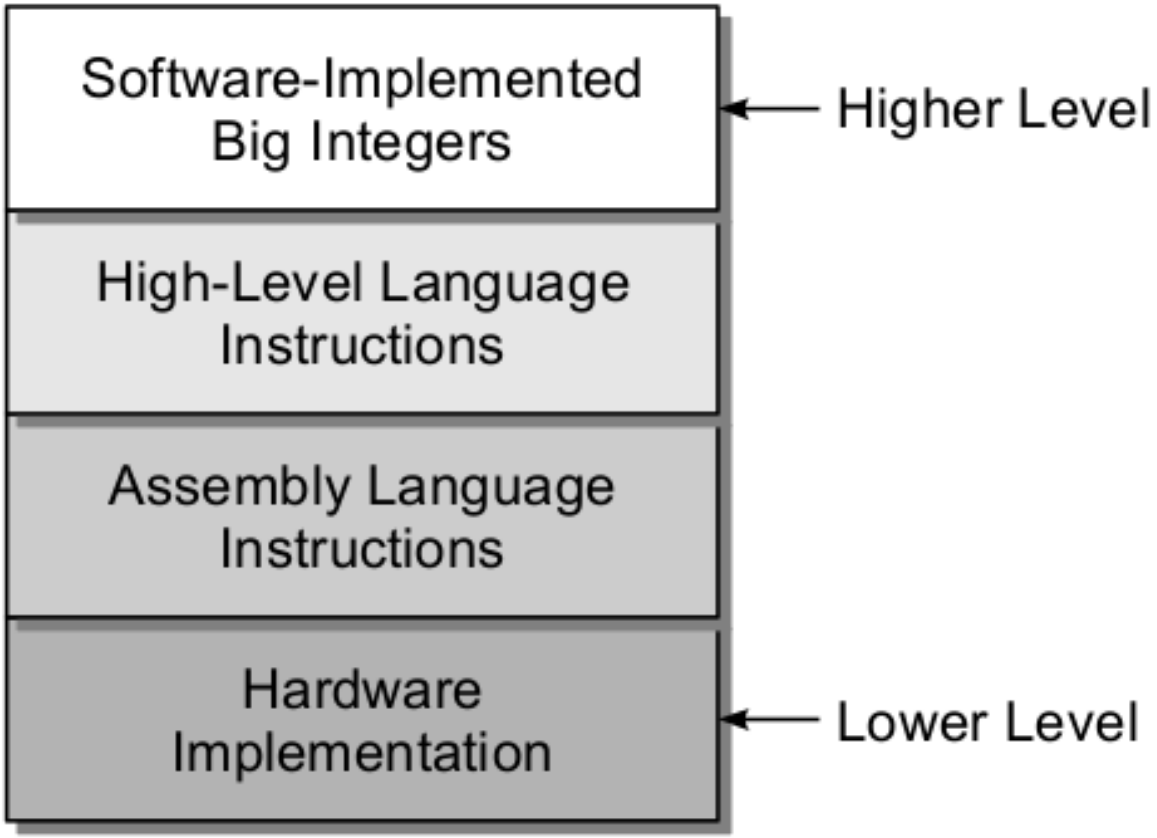
- Muitas vezes, os tipos primitivos oferecidos pelas linguagens de programação não são suficientes para resolver problemas maiores ou mais complexos.
- Portanto, a maioria das linguagens de programação oferece recursos para que os programadores criem os seus próprios tipos de dados.

Abstração

- É um mecanismo para separar as propriedades de um objeto e restringir o foco para o que seja realmente relevante.

Abstração

- Exemplo de diversos níveis de abstração com aritmética de inteiros:



Tipo Abstrato de Dados (TAD)

Tipo Abstrato de Dados (TAD)

- Ou ***Abstract Data Type*** **(ADT)**.

Tipo Abstrato de Dados (TAD)

- Ou ***Abstract Data Type*** **(ADT)**.
- É um tipo de dados definido pelo programador que especifica um conjunto de valores de dados e uma coleção bem definida de operações que podem ser executadas nesses valores.

Tipo Abstrato de Dados (TAD)

- Ou ***Abstract Data Type*** **(ADT)**.
- É um tipo de dados definido pelo programador que especifica um conjunto de valores de dados e uma coleção bem definida de operações que podem ser executadas nesses valores.
- TAD são definidos de maneira independente da sua implementação.

Tipo Abstrato de Dados (TAD)

- Ou ***Abstract Data Type*** **(ADT)**.
- É um tipo de dados definido pelo programador que especifica um conjunto de valores de dados e uma coleção bem definida de operações que podem ser executadas nesses valores.
- TAD são definidos de maneira independente da sua implementação.
- A interação com um TAD é realizado através da sua **interface** ou pelo seu conjunto de funções.

Tipo Abstrato de Dados (TAD)

- Ou ***Abstract Data Type*** **(ADT)**.
- É um tipo de dados definido pelo programador que especifica um conjunto de valores de dados e uma coleção bem definida de operações que podem ser executadas nesses valores.
- TAD são definidos de maneira independente da sua implementação.
- A interação com um TAD é realizado através da sua **interface** ou pelo seu conjunto de funções.
- A consequência disso é o **ocultamento de informação** (*information hiding*).

Tipo Abstrato de Dados (cont.)

- Um TAD é como uma caixa-preta.

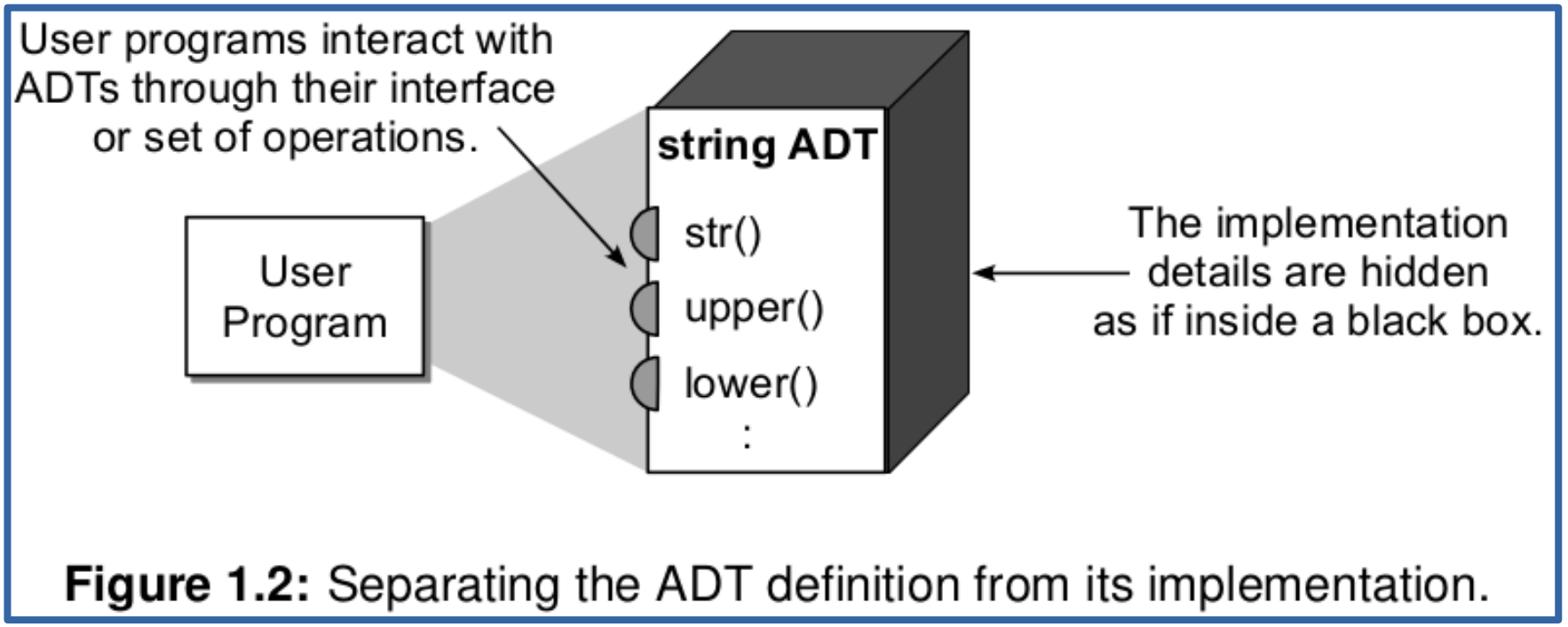


Figure 1.2: Separating the ADT definition from its implementation.

Tipo Abstrato de Dados (cont.)

Tipo Abstrato de Dados (cont.)

- O conjunto de operações pode ser agrupado em quatro categorias:

Tipo Abstrato de Dados (cont.)

- O conjunto de operações pode ser agrupado em quatro categorias:
  - **Construtores (*Constructor*)**: criam e inicializam novas instâncias do TAD.

Tipo Abstrato de Dados (cont.)

- O conjunto de operações pode ser agrupado em quatro categorias:
  - **Construtores (*Constructor*)**: criam e inicializam novas instâncias do TAD.
  - **Acessores (*Accessor*)**: Retorna os dados contidos em uma instância sem modificá-la.

Tipo Abstrato de Dados (cont.)

- O conjunto de operações pode ser agrupado em quatro categorias:
  - **Construtores (*Constructor*)**: criam e inicializam novas instâncias do TAD.
  - **Acessores (*Accessor*)**: Retorna os dados contidos em uma instância sem modificá-la.
  - **Modificadores (*Mutator*)**: Modifica o conteúdo de uma instância de um TAD.

Tipo Abstrato de Dados (cont.)

- O conjunto de operações pode ser agrupado em quatro categorias:
  - **Construtores (*Constructor*)**: criam e inicializam novas instâncias do TAD.
  - **Acessores (*Accessor*)**: Retorna os dados contidos em uma instância sem modificá-la.
  - **Modificadores (*Mutator*)**: Modifica o conteúdo de uma instância de um TAD.
  - **Iteradores**: Processa os dados individuais dos componentes.

Python Construtor

Python Construtor

- Em Python, um construtor é um método especial usado para inicializar as instâncias de uma classe.

Python Construtor

- Em Python, um construtor é um método especial usado para inicializar as instâncias de uma classe.
- Os construtores podem ser de dois tipos:

Python Construtor

- Em Python, um construtor é um método especial usado para inicializar as instâncias de uma classe.
- Os construtores podem ser de dois tipos:
  - Construtor parametrizado.

Python Construtor

- Em Python, um construtor é um método especial usado para inicializar as instâncias de uma classe.
- Os construtores podem ser de dois tipos:
  - Construtor parametrizado.
  - Construtor não-parametrizado.

Python Construtor

- Em Python, um construtor é um método especial usado para inicializar as instâncias de uma classe.
- Os construtores podem ser de dois tipos:
  - Construtor parametrizado.
  - Construtor não-parametrizado.
- O construtor é executado quando nós criamos o objeto de uma classe.

Python Construtor

- Em Python, um construtor é um método especial usado para inicializar as instâncias de uma classe.
- Os construtores podem ser de dois tipos:
  - Construtor parametrizado.
  - Construtor não-parametrizado.
- O construtor é executado quando nós criamos o objeto de uma classe.
- Os construtores também verificam se existem os recursos necessários para que o objeto execute qualquer das suas tarefas.

Python Construtor

Python Construtor

- O método **`__init__`** simula o construtor de uma classe.

Python Construtor

- O método **`__init__`** simula o construtor de uma classe.
- Esse método é chamado quando a classe é instanciada.

Python Construtor

- O método **`__init__`** simula o construtor de uma classe.
- Esse método é chamado quando a classe é instanciada.
- Esse método é comumente usado para inicializar os atributos da classe.

Python Construtor

- O método **`__init__`** simula o construtor de uma classe.
- Esse método é chamado quando a classe é instanciada.
- Esse método é comumente usado para inicializar os atributos da classe.
- Toda classe deve ter um construtor, mesmo se ele simplesmente confiar no construtor padrão (*default*).

Python Construtor

- Exemplo:

```
class Employee:
    def __init__(self,name,id):
        self.id = id;
        self.name = name;
    def display (self):
        print("ID: %d \nName: %s"%(self.id,self.name))

emp1 = Employee("John",101)
emp2 = Employee("David",102)

#accessing display() method to print employee 1 information
emp1.display()

#accessing display() method to print employee 2 information
emp2.display()
```

Python Construtor

- Exemplo (saída):

```
ID: 101
Name: John
ID: 102
Name: David
```

Python Construtor

Python Construtor

- Exemplo: contando o número de objetos de uma classe.

Python Construtor

- Exemplo: contando o número de objetos de uma classe.

```
#SOURCE: https://www.javatpoint.com/python-constructors

class Student:
        count = 0
        def __init__(self):
            Student.count = Student.count + 1

s1=Student()
s2=Student()
s3=Student()

print("The number of students:",Student.count)
```

Python Construtor

- Exemplo: contando o número de objetos de uma classe.

```
#SOURCE: https://www.javatpoint.com/python-constructors

class Student:
        count = 0
        def __init__(self):
            Student.count = Student.count + 1

s1=Student()
s2=Student()
s3=Student()

print("The number of students:",Student.count)
```

```
The number of students: 3
```

Python Construtor

Python Construtor

- Exemplo (não-parametrizado)

Python Construtor

- Exemplo (não-parametrizado)

```
#SOURCE:https://www.javatpoint.com/python-constructors

class Student:
    # Constructor - non parameterized
    def __init__(self):
        print("This is non parametrized constructor")
    def show(self,name):
        print("Hello",name)

student = Student()
student.show("John")
```

Python Construtor

- Exemplo (não-parametrizado)

```
#SOURCE:https://www.javatpoint.com/python-constructors

class Student:
    # Constructor - non parameterized
    def __init__(self):
        print("This is non parametrized constructor")
    def show(self,name):
        print("Hello",name)

student = Student()
student.show("John")
```

```
This is non parametrized constructor
Hello John
```

Python Construtor

Python Construtor

- Exemplo (parametrizado)

Python Construtor

- Exemplo (parametrizado)

```
#SOURCE:https://www.javatpoint.com/python-constructors

class Student:
        # Constructor - parameterized
        def __init__(self, name):
            print("This is parametrized constructor")
            self.name = name
        def show(self):
            print("Hello",self.name)

student = Student("John")
student.show()
```

Python Construtor

- Exemplo (parametrizado)

```
#SOURCE:https://www.javatpoint.com/python-constructors

class Student:
        # Constructor - parameterized
        def __init__(self, name):
            print("This is parametrized constructor")
            self.name = name
        def show(self):
            print("Hello",self.name)

student = Student("John")
student.show()
```

```
This is parametrized constructor
Hello John
```

Python In-built class functions

Python In-built class functions

SN	Function	Description
1	getattr(obj,name,default)	It is used to access the attribute of the object.
2	setattr(obj, name,value)	It is used to set a particular value to the specific attribute of an object.
3	delattr(obj, name)	It is used to delete a specific attribute.
4	hasattr(obj, name)	It returns true if the object contains some specific attribute.

Python In-built class functions

```
class Student:
        def __init__(self,name,id,age):
            self.name = name;
            self.id = id;
            self.age = age

#creates the object of the class Student
s = Student("John",101,22)

#prints the attribute name of the object s
print(getattr(s,'name'))

# reset the value of attribute age to 23
setattr(s,"age",23)

# prints the modified value of age
print(getattr(s,'age'))

# prints true if the student contains the attribute with name id

print(hasattr(s,'id'))
# deletes the attribute age
delattr(s,'age')

# this will give an error since the attribute age has been deleted
print(s.age)
```

Python In-built class functions

- Saída (código anterior):

```
John
23
True
Traceback (most recent call last):
  File "exemplo-builtin-class-functions.py", line 28, in <module>
    print(s.age)
AttributeError: Student instance has no attribute 'age'
```

Python built-in class attributes

SN	Attribute	Description
1	__dict__	It provides the dictionary containing the information about the class namespace.
2	__doc__	It contains a string which has the class documentation
3	__name__	It is used to access the class name.
4	__module__	It is used to access the module in which, this class is defined.
5	__bases__	It contains a tuple including all base classes.

Python built-in class attributes

Python built-in class attributes

```
#SOURCE:https://www.javatpoint.com/python-constructors

class Student:
        def __init__(self,name,id,age):
            self.name = name;
            self.id = id;
            self.age = age
        def display_details(self):
            print("Name:%s, ID:%d, age:%d"%(self.name,self.id))

s = Student("John",101,22)

print(s.__doc__)
print(s.__dict__)
print(s.__module__)
```

Python built-in class attributes

```
#SOURCE:https://www.javatpoint.com/python-constructors

class Student:
        def __init__(self,name,id,age):
            self.name = name;
            self.id = id;
            self.age = age
        def display_details(self):
            print("Name:%s, ID:%d, age:%d"%(self.name,self.id))

s = Student("John",101,22)

print(s.__doc__)
print(s.__dict__)
print(s.__module__)
```

```
None
{'name': 'John', 'id': 101, 'age': 22}
__main__
```

Python Accessor

Python Accessor

- Um método accessor retorna a informação sobre o objeto, mas não muda o estado ou o objeto.

Python Accessor

- Um método accessor retorna a informação sobre o objeto, mas não muda o estado ou o objeto.
- Esse método normalmente é usado com a palavra **get**.

Python Mutator

Python Mutator

- Um método *mutator* é uma função que modifica a variável interna de alguma maneira.

Python Mutator

- Um método *mutator* é uma função que modifica a variável interna de alguma maneira.
- A mais simples forma de um *mutator* é atribuir um novo valor para uma variável.

Python Mutator

- Um método *mutator* é uma função que modifica a variável interna de alguma maneira.
- A mais simples forma de um *mutator* é atribuir um novo valor para uma variável.
- Esse método normalmente é usado com a palavra **set**.

Python Accessor/Mutator

Python Accessor/Mutator

```
#SOURCE: https://www.python-course.eu/python3_properties.php

class P:
    def __init__(self,x):
        self.__x = x

    def get_x(self):
        return self.__x

    def set_x(self, x):
        self.__x = x

p1 = P(42)
p2 = P(4711)
print p1.get_x()

p1.set_x(47)
p1.set_x(p1.get_x()+p2.get_x())
print p1.get_x()
```

Python Accessor/Mutator

```
#SOURCE: https://www.python-course.eu/python3_properties.php

class P:
    def __init__(self,x):
        self.__x = x

    def get_x(self):
        return self.__x

    def set_x(self, x):
        self.__x = x

p1 = P(42)
p2 = P(4711)
print p1.get_x()

p1.set_x(47)
p1.set_x(p1.get_x()+p2.get_x())
print p1.get_x()
```

42

Python Accessor/Mutator

```
#SOURCE: https://www.python-course.eu/python3_properties.php

class P:
    def __init__(self,x):
        self.__x = x

    def get_x(self):
        return self.__x

    def set_x(self, x):
        self.__x = x

p1 = P(42)
p2 = P(4711)
print p1.get_x()

p1.set_x(47)
p1.set_x(p1.get_x()+p2.get_x())
print p1.get_x()
```

42

4758

Python Accessor/Mutator

```
#SOURCE: https://www.python-course.eu/python3_properties.php

class P:
    def __init__(self,x):
        self.__x = x

    def get_x(self):
        return self.__x

    def set_x(self, x):
        self.__x = x

p1 = P(42)
p2 = P(4711)
print p1.get_x()

p1.set_x(47)
p1.set_x(p1.get_x()+p2.get_x())
print p1.get_x()
```

42

encapsulamento

4758

Python Accessor/Mutator

Python Accessor/Mutator

```
#SOURCE: https://www.python-course.eu/python3_properties.php

class P:

    def __init__(self,x):
        self.x = x

p1 = P(42)
p2 = P(4711)
print p1.x

p1.x = 47
p1.x = p1.x + p2.x
print p1.x
```

Python Accessor/Mutator

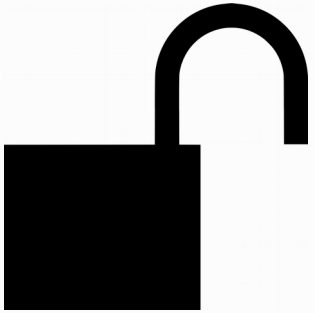
```
#SOURCE: https://www.python-course.eu/python3_properties.php

class P:

    def __init__(self,x):
        self.x = x

p1 = P(42)
p2 = P(4711)
print p1.x

p1.x = 47
p1.x = p1.x + p2.x
print p1.x
```



encapsulamento?

Iterators

Iterators

- Um *iterator* pode ser visualizado como um ponteiro para um *container*, isto é, uma estrutura do tipo lista que pode percorrer sobre todos os elementos deste *container*.

Iterators

- Um *iterator* pode ser visualizado como um ponteiro para um *container*, isto é, uma estrutura do tipo lista que pode percorrer sobre todos os elementos deste *container*.
- Exemplo:

```
cities = ["Paris","Berlin","Frankfurt"]

for location in cities:

  print("location: " + location)
```

Tipo Abstrato de Dados (cont.)

Tipo Abstrato de Dados (cont.)

- Existem algumas vantagens no uso de TADs:

Tipo Abstrato de Dados (cont.)

- Existem algumas vantagens no uso de TADs:
  - Foco na resolução do problema ao invés de se preocupar com detalhes de implementação.

Tipo Abstrato de Dados (cont.)

- Existem algumas vantagens no uso de TADs:
  - Foco na resolução do problema ao invés de se preocupar com detalhes de implementação.
  - A implementação do TAD pode ser modificada sem ter a necessidade de modificar o programa que utiliza o TAD.

Tipo Abstrato de Dados (cont.)

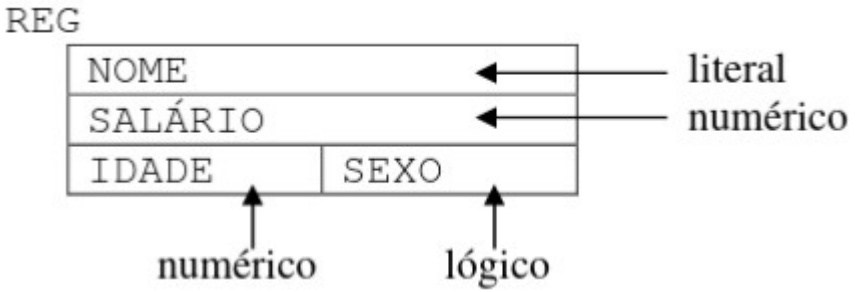
- Existem algumas vantagens no uso de TADs:
  - Foco na resolução do problema ao invés de se preocupar com detalhes de implementação.
  - A implementação do TAD pode ser modificada sem ter a necessidade de modificar o programa que utiliza o TAD.
  - É mais fácil gerenciar e dividir programas grandes (reais) em módulos menores.

Referências

- https://www.javatpoint.com/python-constructors

Exercícios

- Explique a vantagem do uso de tipos abstratos de dados. Dê um exemplo.
- Crie um TAD em Python para o elemento abaixo:



Exercícios

- Um número complexo é representado como x + i.y, onde i2 = -1, sendo x a sua parte real e y a sua parte imaginária. Ambas são representadas por números reais. Crie um TAD em Python que represente os números complexos com as seguintes funções:
  - Criar um número complexo.
  - Destruir um número complexo.
  - Soma de dois números complexos.
  - Subtração de dois números complexos.

Exercícios

- Crie um TAD que represente um ponto em um plano 2D, com métodos para definir as coordenadas e outro para calcular a distância até outro ponto.
- Implemente um TAD que permita adicionar, remover, buscar e listar contatos. Cada contato pode ter nome, telefone e email.
- Crie um TAD para um produto em uma loja e demonstre como usar métodos acessores (getters) e modificadores (setters) para manipular os atributos do produto de forma controlada.

Exercícios

- Considere o código abaixo e depois responda ao que se pede:

python Copy code

```python
class Relogio:
    def __init__(self, hora, minuto, segundo):
        self.hora = hora
        self.minuto = minuto
        self.segundo = segundo

    def set_time(self, hora, minuto, segundo):
        self.hora = hora
        self.minuto = minuto
        self.segundo = segundo

    def get_time(self):
        return f"{self.hora:02}:{self.minuto:02}:{self.segundo:02}"

# Exemplo de uso
relogio = Relogio(14, 30, 25)
print(relogio.get_time())
relogio.set_time(16, 45, 30)
print(relogio.get_time())
```

Exercícios

- Perguntas:
  - O que acontece se você tentar acessar diretamente um atributo do objeto relogio, como relogio.hora?
  - Modifique a classe para impedir o acesso direto aos atributos hora, minuto, e segundo de fora da classe.
  - Adicione um método que avança o tempo em um segundo e ajusta automaticamente minutos e horas se necessário.