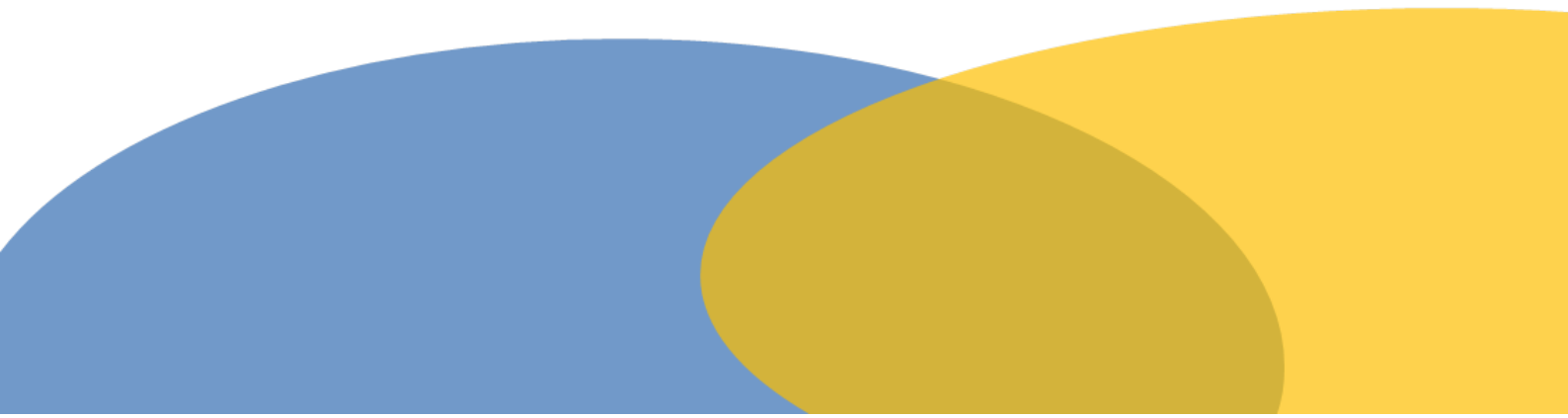
Introdução à Linguagem Python

Paradigmas de Linguagens de Programação

Ricardo Willian Pontes da Silva
Ausberto S. Castro Vera

12 de outubro de 2022



UENF - UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE DARCY RIBEIRO

CCT - CENTRO DE CIÊNCIA E TECNOLOGIA
LCMAT - LABORATÓRIO DE MATEMÁTICAS
CC - CURSO DE CIÊNCIA DA COMPUTAÇÃO

Primeira edição, Maio 2019

Sumário

1. Introdução

A linguagem de Programação Python, criada pelo holandês Guido Van Rossum em meados das décadas de 1980 à 1990, se caracteriza por ser do tipo *Very High Level Languagens* (VHLL), ou seja, isso torna a linguagem em alto nível de abstração, porém não voltada apenas para a programação profissional, mas também se destacando em instituições de ensino e por indivíduos autodidatas que desejam ingressar no Python como primeira linguagem. Desta forma, Python também se destaca em detrimento das demais linguagens de programação existentes no mercado pela simplicidade em sua sintaxe, código aberto e também por ser executável em multi-plataformas.

> Com a evolução da programação, os paradigmas das linguagens também foram se modificando com relação ao tempo, com a linguagem Python não foi diferente, tendo como base o paradigma orientado à objetos, mas também ainda permitindo a criação de códigos na forma procedural, abrangendo diferentes tipos de aplicações (de grandes e pequenas proporções) e desenvolvedores (Profissionais do nível básico ao avançado). Com isso, Python possibilita a migração de um paradigma para outro de forma natural e espontânea por parte do programador.

1.1 Aspectos históricos da linguagem Python

Python surgiu como uma possível solução para as deficiências das linguagens de programação mais aquecidas da época, como C, tendo como filosofia principal a fácil aplicação e intuitiva.

A seguir, menciona-se alguns aspectos históricos da linguagem Python, baseados em[Per16], [Sev15],[Lab]:

- Criado por Guido van Rossum, um holandês de aproximadamente 26 anos em 1982, quando trabalhava no CWI (Centrum Wiskunde & Informatica), em Amsterdã, Holanda.
- Após pensar na linguagem de programação, sua nomeação veio com base no padrão exercido pela organização que seu criador atuava, que se dava por nomenclaturas de animais. Desta forma, Van Rossum, que era fã de um seriado de comédia da BBC *Monty Python's Flying Circus* nomeou-a como Python. .
- Python 0.9.0 foi lançado em 1991. Esta versão incluia manipulação de exceções, classes,

listas e strings. Também foram inclídos alguns aspectos de programação funcional: lambda, maps flitros e reduce.
- Em 1995, Guido continuou se trabalho sobre Python na Corporation for National Research Initiatives (CNRI) em Reston, Virginia, USA.
- Python 1.6 foi lançado no CNRI em
- No ano 2000, Guido e a equipe de desenvolvimento principal do Python foram para BeOpen.com para formar a equipe BeOpen PythonLabs.
- Python 2.0 foi lançado no ano 2000
- Em 2001, foi formada a PSF (Python Software Foundation), uma organização sem fins lucrativos que mantém e coordena o uso da linguagem. Atualmente, o PSF é suportado por grandes empresas como Google, Microsoft e Globo.com, que também utilizam Python em seus sistemas.
- Python 3.o foi lançado em dezembro de 2008

Figura 1.1: Logo da Linguagem Python

Fonte: (VRDJ95)

1.2 Áreas de Aplicação da Linguagem

A linguagem Python, diferente da grande maioria em uso atualmente, não possui uma aplicação em específico, podendo ser encontrada no desenvolvimento *Backend/Frontend* como são divididas atualmente áreas mais fomentadas da computação, mas também sendo utilizada em meios como processamento de imagem, aplicações *Mobile, Data Science* e inúmeras outras.
Vale destacar também que, como a linguagem de programação Python é muito versátil, diversas companhias utilizam de sua estrutura para aplicar em seus serviços que serão fornecidos ao cliente. Podemos citar empresas que utilizam-se da linguagem Python: Dropbox, Yahoo, Intel, Cisco, HP , IBM e etc. Como citado por [dSS19]
A linguagem Python também pode ser integrada com diversas outras linguagens de programação, como Java, JavaScript e C. Desta forma, abrangendo ainda mais o seu uso e o perfil do profissional ao qual irá ser responsável pela aplicação.

1.2.1 Big Data

Com o avanço da tecnologia e a conectividade em praticamente todos os meios populacionais, é necessário um estudo e análise dos dados gerados por essa conexão. A área da tecnologia responsável por essa manipulação é chamada de análise de dados, que se tornará ainda mais necessária para que haja tomadas de decisões assertivas.
Por ser uma linguagem fortemente prototipada, de alto nível, orientada a objetos e dinâmica, o Python é uma das mais famosas linguagens para se trabalhar com manipulação de dados e desenvolvimento científico atuantes no mercado.[McK19]
Outro fator que contribui para o Python ser largamente utilizado nesse meio de aplicação quantitativa e analítica é a grande quantidade de bibliotecas existentes que auxiliam desde o

profissional mais iniciante ao mais experiente, com arquitetura robusta para suportar grandes aplicações.Como algumas dessas bibliotecas, podemos citar, NumPy,Pandas, Matplotlib dentre diversas outras

1.2.2 Orientação a objetos

Python é classificada como uma linguagem orientada a objetos, porém também é permissiva ao desenvolvimento de forma imperativa e funcional, ou seja, dizemos que Python é uma linguagem de multi-paradigmas. O paradigma orientado a objeto é baseado no conceito de classes, objetos,métodos e atributos, que visam representar o mundo real [Bar19].
Nesta linguagem de programação, tudo o que existe é um objeto. Atributos, classes, métodos, números e tudo o que existe em uma linguagem mesmo que não necessariamente orientada a objetos, se torna um objeto ao contrário da programação orientada a procedimentos.
Desta forma, a Orientação a Objetos na linguagem Python proporciona uma otimização do tempo, maior qualidade nos projetos e maior clareza de entendimento por parte do programador a qual está desenvolvendo a aplicação. Tais pontos positivos se dão uma vez que a estrutura da Python visa retratar o mais próximo possível de como as coisas se agrupam na vida real.

1.2.3 Processamento de imagens

Pela sua sintaxe de simples entendimento, robustez para grades aplicações e agilidade, o Python é altamente utilizado na área da computação responsável por processar e extrair informações de imagens. Esse processo se faz necessário nos dias atuais uma vez que a automação e inteligência dos aparelhos está cada vez mais presente no cotidiano, como, redares inteligentes, seleção de alimentos, dentre diversas outras.
Para realizar essa operações e análises, Python conta com uma grande quantidade de bibliotecas funcionais e uma significativa comunidade para auxiliar o programador. Podemos citar bibliotecas como, NumPy (*Numeric Python*), Scypy (*Cientific Python*), Matplotlib. Ambas têm como função principal operar funções matemáticas de maneira ágil e eficiente, como matrizes e vetores. Vale destacar também que, tais áreas que envolvem tomadas de decisão como processamento de imagem, inteligência artificial e etc, são tratadas como a nova fase da computação, onde aparelhos serão capazes de se assemelhar com os seres humanos em atividades rotineiras. Desta forma, podemos definir a linguagem Python como sendo uma linguagem do futuro e que possibilitará ao programador se manter atualizado[Bar19].

2. Conceitos básicos da Linguagem Python

Neste capítulo será apresentado conceitos básicos da linguagem Python, como, seus tipos primitivos de variáveis e constantes presentes na linguagem, suas estruturas, sua sintaxe e semântica e também seu o uso de pacotes disponíveis para a construção de códigos. Os livros básicos para o estudo da Linguagem Python são:[Man18] ,[Sum13], [Gut15], [Per16]

Considerando que a linguagem Python ([Man18]) é definida como sendo pseudocódigo executável pela sua semelhança com a facilidade de entendimento de algoritmos voltados para a aprendizagem. Desta forma, nos sentiremos familiarizados com a construção de seus códigos, ou então não ocorrerá grandes dificuldades de serem entendidos.

2.1 Variáveis e constantes

Antes de entendermos o que se refere a parte prática da aplicação de variáveis e constantes na linguagem Python, vale ressaltar uma nota que já foi expresso anteriormente, que se trata do fato de que nesta linguagem, tudo é classificado como objeto. Desta forma, conceitos anteriormente conhecidos de linguagens anteriores não serão mais utilizados, como a direta referência de memoria ao ocorrer a criação e alocação de variáveis.
Com isso, é dito que a linguagem Python é dinâmicamente tipada, o que refere ao fato de não se precisar especificar o seu tipo primitivo "tamanho que ocuparia na memória".

2.2 Tipos de Dados Básicos

De acordo com, [Men16], em Python temos variáveis de tipos primitivos como *int*, que alocará números inteiro entre -2.147.483.648 a 2.147.483.647, ou seja, esse tipo de variável tem como tamanho na memória 32 bits. Temos também o tipo *float*, que tem como capacidade armazenar números facionários entre $1.7 * 10^{-8}$ a $3.4 * 10^{8}$. E por fim, temos também o tipo primitivo *string*, que terá seu tamanho na memória variável de acordo com sua alocação.

2.2.1 Inteiro (int)

Dizemos que um valor atribuído a uma variável é do tipo *int* se e somente se, o mesmo não houver pontos flutuantes. Abaixo está sendo representado um exemplo de atribuição na linguagem Python para um melhor entendimento.

```
>>> #atribuicao simples em Python (int)
>>> numero = 2
>>> numero += 4
O valor final da variavel 'numero' sera 6, uma vez que
a mesma recebeu duas atribuicoes.
```

Assim como em outras linguagens de programação, é possível fazer mudança de tipos de um valor já existente de uma variável para outra. Por exemplo, utilizar do chamado *cast* para atribuir a uma variável do tipo *int*, um valor que como já visto anteriormente, não se encaixaria nesta categoria. Abaixo será apresentado exemplos de *casting* na linguagem Python.

```
>>> #atribuicao com casting na linguagem Python
>>> x = int(3) O valor de x sera 3
>>> y = int(3.5) O valor de y sera 3
>>> z = int("3") O valor de z sera 3
```

2.2.2 Ponto flutuante (float)

Assim como visto anteriormente, uma variável pode ser considerada do tipo *float* se a mesma conter números facionários, ou seja, contendo números de ponto flutuante. Abaixo será apresentado um pequeno trecho de código contendo um exemplo de atribuição para um melhor entendimento.

```
>>> #atribuicao simples em Python (float)
>>> numero = 2.3
>>> valor = 2.1
>>> total = numero + valor
O valor final da variavel 'total' sera 4.4, uma vez que
a mesma recebeu a atribuicao da variavel 'numero'
que contem 2.3 e da variavel 'valor' 2.1.
```

A linguagem Python também conta com uma funcionalidade de retorno do tipo de uma variável existente. Essa prática tende a ajudar o programador ao decorrer do código em atividades que necessitem de uma tomada de decisão por exemplo. Abaixo será apresentado um trecho de código com o objetivo de elucidar o conteúdo.

```
>>> #Retorno de um tipo na linguagem Python (float)
>>> x = 2.15.
>>> y = 3.0
>>> z = -0.32

print(type(x))
print(type(y))
print(type(z))
```

2.2.3 String

Um string é uma sequência de caracteres considerado como um item de dado simples. Para Python, um string é um array de caracteres ou qualquer grupo de caracteres escritos entre doble aspas ou aspas simples. Por exemplo,

```
>>> #usando aspas simples
>>> pyStr1 = 'Brasil'
>>> print (pyStr1)  Brasil
>>> #usando aspas duplas
>>> pyStr2 = "Oi, tudo bem?"
>>> print (pyStr2)
Oi, tudo bem?
```

- *Concatenação de strings*
  Strings podem ser concatenadas utilizando o operador +, e o seu comprimento pode ser calculado utilizado o operador `len(string)`

```
>>> # concatenando 2 strings
>>> pyStr = "Brasil" + " verde amarelo"
>>> print (pyStr)
Brasil verde amarelo
>>> print (len(pyStr))
20
```

- *Operador de indexação*
  Qualquer caracter de um string o sequência de caracteres pode ser obtido utilizando o operador de indexação []. Existem duas formas de indexar em Python, os caracteres de um string:

  Index com inteiros positivos indexando a partir da esquerda começando com 0 e onde 0 é o index do primeiro caracter da sequência

  Index com inteiros negativos indexando a partir da direita começando com -1, e onde -1 é o último elemento da sequência, -2 é o penúltimo elemento da sequência, e assim sucessivamente.

```
>>> # Indexando strings
>>> pyStr = "Programando"
>>> print (len(pyStr))
11
>>> print (pyStr)
Brasil verde amarelo
```

- *Operador de Fatias*
  O operador de acesso a itens (caracteres individuais) também pode ser utilizado como operador de fatias, para extrair uma fatia inteira (subsequência) de caracteres de um string. O operador de Fatias possui três sintaxes:
  `seq[ inicio ]`
  `seq[ inicio : fim ]`
  `seq[ início : fim : step ]`
  onde `início`, `fim` e `step` são números inteiros.

```
>>> # Indexando strings
>>> pyStr = "Programando Python"
>>> print (len(pyStr))
11
>>> print (pyStr)
Brasil verde amarelo
```

2.3 Tipos de Dados de Coleção

Como visto antes, variáveis só poderiam alocar em seu espaço de memória um único tipo primitivo apenas, porém, neste tópico será apresentado um conceito de dados de coleção onde será possível armazenar N itens dentro de uma única variável. Como principais tipos de coleção existentes na linguagens Python temos: Tuplas, Sets e dicionários, que serão apresentados abaixo.

2.3.1 Tipos Sequenciais (tuplas)

Em Python, utilizamos as chamadas Tuplas para armazenar vários itens em uma única variável. Tupla é classificado como um dos 4 tipos de dados internos usados para armazenar coleções de dados. Uma tupla é uma coleção ordenada e imutável, a sintaxe utilizada para a implementação das Tuplas são os colchetes.

Os itens de uma tupla são classificados como, ordenados, imutáveis e permissivos a elementos duplos além de serem indexados, ou seja, cada posição é referenciada por um índice que se inicia na posição [0], [1] e etc. Abaixo veremos um exemplo da utilização das tuplas.

```
>>> # Uso de Tuplas na linguagem Python
>>> ExemploTupla = ("uva", "banana", "mel",
"uva")
>>> print(ExemploTupla)
```

As tuplas também podem receber outros tipos de dados primitivos como os que já foram citados anteriormente, como, *int* e *float*. Também é possível saber quantos elementos estão contidos em uma Tupla. Abaixo será expresso em um trecho de código tais utilizações.

```
>>> # Uso de Tuplas com diversos tipos primitivos na
    #linguagem Python
>>> tupla1 = ("uva", "banana", "mel")
>>> tupla2 = (10, -2, 8, 1, 3)
>>> tupla3 = (True, False, False)
>>> tupla4 = ("Linguagem Python", 2.4, True, 40, "Oi")
>>> print(len(tupla4))
```

2.3.2 Tipos Conjunto (set)

Assim como as Tuplas, utilizamos os chamados *set* para armazenar vários itens em um única variável. Porém, o grande diferencial do tipo de dados *set* é por ser uma coleção não ordenada, imutável e não indexada, vale destacar também que o tipo de dados *set* não aceita dados duplicados. Abaixo será ilustrado em um trecho de código a utilização básica do comando *set*, sendo armazenado dados homogêneos e heterogêneos.

```
>>> # Uso de sets com diversos tipos primitivos na
#linguagem Python
set1 = {"uva", "banana", "mel"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}
set4 = {"Linguagem Python", 2.4, False, 40, "Ola"}
```

Algumas particularidades do tipo de dado de coleção *set* é que como já citado anteriormente, caso já criado, não é possível alterar seus dados, porém é possível criar novos itens e também excluir dados existentes. Vale a ressalva também que o *set* não é referenciado por um índice, logo é possível que os itens apareçam em uma ordem diferente toda vez que utilizados.

2.3.3 Tipos Mapeamento (dicionário)

Por fim, chegamos no tipo de dado de coleção dicionário, que se caracteriza por ser uma coleção de dados ordenada, mutável e que não permite duplicatas.Assim como os demais, o dicionário é permissivo que seja alocados diferentes tipos de dados primitivos em seu corpo de código. A estrutura de um dicionário é delimitada por chaves que tendem a armazenar os dados inseridos, como mostrado no trecho abaixo.

```
        >>> # Uso de dicionarios com diversos tipos primitivos
        # na linguagem Python
        dicionario = {
        "marca": "Ford",
        "eletrico": False,
        "ano": 1964,
        "cor": ["vermelho", "branco", "azul"]
        }
```

2.4 Estrutura de Controle e Estruturas de repetição

Assim como em diversas outras linguagens de programação, o Python também conta com estruturas de controle em sua sintaxe. Basicamente, uma estrutura de controle tem o objetivo de executar um determinado trecho de código se baseando em uma condição pré definida. Já as estruturas de repetição, executam um trecho de código N vezes até que sua condição seja alcançada. Neste tópico será apresentado o comando *if*,*For* e também *while*, que serão utilizados juntamente com operadores lógicos que são, == 'igual', != 'diferente', < 'maior que', > 'menor que', <= / >= 'maior/menor ou igual' em suas condições.

2.4.1 O comando IF

O comando *if* possui a sua estrutura como sendo uma condição logo em seguida, uma declaração. Veremos um exemplo básico de sua utilização na linguagem Python.

```
        >>> # Uso basico de um if na linguagem Python
        if 30 > 10:
                print("30 e maior que 10")
```

Também é possível encadear N *if* para executar trechos de código. Vale destacar também que na linguagem Python, a identação é de extrema importância, uma vez que é desta forma que o interpretador do Python compreende sua estrutura. Abaixo será apresentado uma utilização prática do chamado *elif*.

```
        >>> # Uso basico de um if encadeado na linguagem Python
a = True
b = False
if a == b:
        print(" A e igual a B")
elif a != b:
        print("A nao e igual a B")
```

2.4.2 Laço FOR

Em Python a utilização do loop *for* possui algumas divergências do uso da mesma palavra reservada em outras linguagens de programação, que em sua maioria funciona como um método iterador,

porém no Python, um loop *for* é majoritariamente utilizado em juntamente com coleções ou simplesmente uma *string* comum. Abaixo veremos alguns exemplos da utilização do *for* na linguagem Python.

```
        >>> # Uso basico de for na linguagem Python
        frutas = ["uva", "banana", "amora"]
        for x in frutas:
                print(x)
        #Este trecho de codigo tem como objetivo retornar
        #todos elementos dessa colecao
```

É possível também utilizar de estruturas de repetição com outras estruturas, como a de controle por exemplo ou mesmo. Tal funcionalidade especifica um bloco de código a ser executado quando o ciclo *for* for finalizado. Abaixo veremos um pequeno trecho de código da a utilização de duas estruturas em um contexto prático.

```
        >>> # Uso basico de for com estruturas de controle na
            # linguagem Python
for numero in range(11):
        print(numero)
else:
        print("Ciclo finalizado!")
```

2.4.3 Laço WHILE

Outra estrutura de repetição existente na linguagem Python é o ciclo *while*, que se assemelha bastante ao que é largamente utilizado em outras linguagens de programação. Com o ciclo *while* é possível a execução de um trecho de instruções N vezes até a condição deixar de ser satisfeita. Abaixo veremos uma simples aplicação do ciclo *while* na linguagem Python.

```
        >>> # Uso basico de ciclo while na linguagem Python
contador = 1
while i <= 10:
        print(contador)
        contador += 1
```

Vale ressaltar que, diferente do ciclo *for* já visto anteriormente, o ciclo *while* necessita de uma variável de controle que seja alterada na lógica do programa, evitando assim o que se entende por ciclo infinito, onde este trecho de código nunca encerará.

2.5 Módulos e pacotes

A linguagem Python possui diversas ferramentas com o objetivo de auxiliar na construção de códigos. Algumas delas são as possibilidades da reutilização de uma estrutura de código já pronta na linguagem. Essas ferramentas possuem as nomenclaturas de Módulos e Pacotes e neste capítulo será debruçado nos conceitos e utilizações destas ferramentas.

2.5.1 Módulos

Os módulos na linguagem Python são programas contendo todos os conceitos anteriormente vistos e também podendo haver a arquitetura orientada a objetos, como classes a objetos. Estes pequenos programas tendem a fornecer alguma funcionalidade que ficará disponível sempre que for requerida.
A linguagem Python também permite que seja criados módulos do zero e assim utilizados ao

decorrer da necessidade. Abaixo veremos uma pequena demostração de como criar módulos e utiliza-los.

```
>>> # Cricao de modulos na linguagem Python
def greeting(nome):
print("Oi, " + nome)
# Para utilizar um modulo e preciso salvar-lo como a
# extensao .py
```

Para a utilização de um módulo já existente basta invocá-lo com a palavra reservada *import* seguida do nome salvo do módulo. Abaixo será apresentado como ocorre este evento na prática utilizando a linguagem Python.

```
>>> # Utilizacao de modulos na linguagem Python
import olaModulo
olaModulo.greeting("Ausberto")
```

2.5.2 Pacotes

Por fim, temos também disponível na linguagem Python os chamados pacotes, que se caracterizam por ser um agregado de módulos. Sua utilização ocorre em diversas áreas como científica, arquiteturas, jogos, plotagem de gráficos dentre diversos outros. Alguns dos principais pacotes existentes são pandas, numpy, matplotlib etc. Abaixo veremos como utilizar pacotes na linguagem Python.

```
>>> # Utilizacao de pacotes na linguagem Python
>>> pip isntall 'nomedopacote'
```

Vale destacar também que é possível criar pacotes assim como módulos e utiliza-los da mesma maneira dos já existentes.

3. Programação Orientada a Objetos com Python

O paradigma Orientado a Objetos é um conjunto de padrões exercidos de modo a solucionar um determinado problema, classificando-o como um objeto do mundo real. Como já citado anteriormente no tópico "Orientação a objetos"na página 7 baseado em [Bar19] , em Python, toda e qualquer construção da linguagem é considerada um objeto. Desta forma, mesmo quando não se usa o paradigma POO, ainda assim se utiliza de objetos.

3.1 Classes

Como citado por [Per16],em Python, uma classe pode ser entendida como um "agrupamento"de métodos e atributos de um determinado objeto. Vale ressaltar também que a criação da classe deverá ocorrer antes da criação do seu respectivo objeto. Abaixo veremos o exemplo da utilização de uma classe na linguagem Python.

```
class MinhaClasse:

 num = 5
```

3.2 Objetos

Desta forma, seguindo a mesma lógica de classe, que busca implementar o mundo real e suas soluções de problemas no conceito de algoritmos computacionais, os objetos podem ser definidos como qualquer existência abstrata de modo que possua comportamentos e características. Abaixo será mostrado um exemplo prático da criação de um objeto na linguagem Python.

```
obj = MinhaClasse()
soma = obj.num + 10
print(soma)
```

Outra funcionalidade das classes contida na linguagem Python é o que se entende como construtor de uma classe no conceito de POO. Esse comando é utilizado para atribuir valores nos determinados

atributos que o objeto está modificando. Abaixo segue um pequeno exemplo de como utilizar a palavra reservada init().

```
class Pessoa:
def __init__(self, nome, idade):
self.nome = nome
self.idade = idade

Joao = Pessoa("Joao", 25)
```

3.3 Operadores ou Métodos

Em suma, podemos classificar um método como sendo um uma função ligada a um objeto instância. Os métodos podem ser de classes definidas pelo próprio usuário, mas também métodos já nativos da própria linguagem. Para se declarar um método em Python, é necessário utilizar da palavra reservada def, seguida de parênteses, que possuem a função de agrupar os parâmetros e por fim, o uso também da palavra reservada init, a qual já foi detalhada anteriormente. Abaixo será apresentado um pequeno exemplo da declaração de um método na linguagem Python.

```
class Pessoa:
def __init__(self, nome, idade):
self.nome = nome
self.idade = idade

def setNome(self, nome):
self.nome = nome

def setIdade(self, idade):
self.idade = idade
```

Vale destacar também que, há uma ligeira diferença entre métodos e funções na linguagem Python. Como já citado no parágrafo anterior, um método está ligado a um objeto, que por sua vez está ligado em uma classe. Já uma função é não está definida dentro de uma classe, portanto, não pertence a um objeto de instância.

3.4 Herança

No conceito do paradigma orientado a objetos, herança é a capacidade pela qual uma linguagem tem de estender funcionalidades de uma classe, ou seja, podemos entender a herança como sendo um tipo de relacionamento entre classes. O uso de herança se faz necessário uma vez que possibilita a reutilização de códigos e também representa de maneira satisfatória o conceito do mundo real em algoritmos computacionais. Abaixo será expressado um simples exemplo da utilização de herança na linguagem Python.

```
class Pessoa:
        def __init__(self, nome, idade):
        self.nome = nome
                self.idade = idade
```

```
class Estudante(Pessoa):
        pass

Ricardo = Estudante ('Ricardo Willian Pontes da Silva',21)
```

Vale destacar o uso da palavra reservada pass na classe Estudante que está estendendo a classe Pessoa,que tem como objetivo herdar atributos e métodos na classe mãe sem incluir seus específicos.

3.5 Estudo de Caso:

De acordo com [Per16], um estudo de caso se define pela análise e tratamento de dados de um determinado assunto, de modo a ser possível entender fenômenos presentes. Na programação o estudo de caso se dá como uma espécie de análise linha por linha de algum determinado trecho de código. Neste capítulo será feita um estudo de caso de um algoritmo na linguagem Python que tem como objetivo a criação e leitura de um determinado arquivo.
Antes de se iniciar a análise de código, é necessário entender alguns detalhes da sintaxe da linguagem Python.

```
'r'  : Leitura de dados
'w'  : Escrita sobreposta de dados ja existentes
'x'  : Escrita se e somente se nao houver dados pre existentes
'a'  : Escrita no final do arquivo
open : Abre um arquivo
write: Insere uma string no arquivo
writelines: intera o texto atraves de um objeto
```

Esses comandos constantemente serão utilizados juntos. Abaixo será expresso um trecho de código para facilitar seu entendimento.

```
file = open("apresentacao", "a")
file.write("nome: Ricardo Willian Pontes da Silva\n")
```

Como já explicado anteriormente, essas duas linhas de código tem como objetivo a criação do arquivo 'apresentacao' e adicionar uma string no mesmo. Neste caso será adicionado um nome.

```
informacoes = list ()
informacoes.append("Idade: 21\n")
informacoes.append("Sexo: Masculino\n")

file.writelines(informacoes)
```

Em um arquivo já criado, o comando writelines inclui informações contidas em estruturas como listas, dicionários e etc. Com isso, no trecho de código acima foi criada uma lista de informações como idade e sexo, e assim foi incrementado no arquivo desejado.

Referências Bibliográficas

[Bar19] Felipe Barelli. Introducao a visao computacional uma abordagem pratica com python e opencv. *Casa do codigo*, 2019. Citado 2 vezes nas páginas 7 e 17.

[dSS19] Rogério Oliveira da Silva and Igor Rodrigues Sousa Silva. *Linguagem de Programacao Python*, volume 10. 2019. Citado na página 6.

[Gut15] John V. Guttag. *Introdução à Computação e Programação Usando Python*. Infopress Nova Mídia, São Paulo, edição revista e ampliada edition, 2015. Citado na página 9.

[Lab] Josue Labaki. *Introducao a Python Modulo A*. Citado na página 5.

[Man18] José Augusto NG Manzano. *Introdução à linguagem Python*. Novatec Editora, 2018. Citado na página 9.

[McK19] Wes McKinney. *Python para analise de dados*. Novatec Editora, May 2019. Citado na página 6.

[Men16] Nilo Ney Coutinho Menezes. *Introdução à programação com Python–2ª edição: Algoritmos e lógica de programação para iniciantes*. Novatec Editora, 2016. Citado na página 9.

[Per16] Ljubomir Perkovic. *Introdução à Computação usando Python: un foco no desenvolvimento de Aplicações*. LTC Livros Técnicos e Científicos Editora Ltda, Rio de Janeiro, 2016. Citado 4 vezes nas páginas 5, 9, 17 e 19.

[Sev15] C. Severance. Guido van Rossum: The Early Years of Python. *Computer*, 48(2):7–9, Feb 2015. Citado na página 5.

[Sum13] Mark Summerfield. *Programação em Python 3 - Uma Introdução Completa à Linguagem Python*. Biblioteca do Programador. Alta Books Editora, Rio de Janeiro, 2013. Citado na página 9.

[VRDJ95] Guido Van Rossum and Fred L Drake Jr. *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, Scotts Valley, CA, 1995. Citado na página 6.

Disciplina: *Paradigmas de Linguagens de Programação 2022*
Linguagem: *Python*
Aluno: *Ricardo Willian Pontes da Silva*

Ficha de avaliação:

Aspectos de avaliação (requisitos mínimos)	**Pontos**
Introdução (Máximo: 01 pontos) • Aspectos históricos • Áreas de Aplicação da linguagem	
Elementos básicos da linguagem (Máximo: 01 pontos) • Sintaxe (variáveis, constantes, comandos, operações, etc.) • Cada elemento com exemplos (código e execução)	
Aspectos Avançados da linguagem (Máximo: 2,0 pontos) • Sintaxe (variáveis, constantes, comandos, operações, etc.) • Cada elemento com exemplos (código e execução) • Exemplos com fonte diferenciada (listing)	
Mínimo 5 Aplicações completas - Aplicações (Máximo : 2,0 pontos) • Uso de rotinas-funções-procedimentos, E/S formatadas • Uma Calculadora • Gráficos • Algoritmo QuickSort • Outra aplicação • Outras aplicações ...	
Ferramentas (compiladores, interpretadores, etc.) (Máximo : 1,0 pontos) • Ferramentas utilizadas nos exemplos: pelo menos DUAS • Descrição de Ferramentas existentes: máximo 5 • Mostrar as telas dos exemplos junto ao compilador-interpretador • Mostrar as telas dos resultados com o uso das ferramentas • Descrição das ferramentas (autor, versão, homepage, tipo, etc.)	
Organização do trabalho (Máximo: 01 ponto) • Conteúdo, Historia, Seções, gráficos, exemplos, conclusões, bibliografia • Cada elemento com exemplos (código e execução, ferramenta, nome do aluno)	
Uso de Bibliografia (Máximo: 01 ponto) • Livros: pelo menos 3 • Artigos científicos: pelo menos 3 (IEEE Xplore, ACM Library) • Todas as Referências dentro do texto, tipo [ABC 04] • Evite Referências da Internet	
Conceito do Professor (Opcional: 01 ponto)	
Nota Final do trabalho:	

Observação: Requisitos mínimos significa a *metade* dos pontos