Pontifícia Universidade Católica de Minas Gerais
ICEI — Instituto de Ciências Exatas e Informática
Ciência da Computação

Iyan Lucas Duarte Marques, Samir do Amorim Cambraia, Guilherme Cosso

PYTHON

Trabalho de Linguagens de Programação

Belo Horizonte
2021

Iyan Lucas Duarte Marques, Samir do Amorim Cambraia, Guilherme Cosso

PYTHON

Trabalho de Linguagens de Programação apresentado ao professor da matéria, como requisito parcial para a conclusão da matéria de Linguagens de Programação.

Belo Horizonte
2021

SUMÁRIO

1 INTRODUÇÃO

Python é uma linguagem de programação de alto nível, interpretada de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Atualmente, possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation. Apesar de várias partes da linguagem possuírem padrões e especificações formais, a linguagem, como um todo, não é formalmente especificada. O padrão de facto é a implementação CPython.[1]

Parte da cultura da linguagem gira ao redor de The Zen of Python, um poema que faz parte do documento "PEP 20 (The Zen of Python)", escrito pelo programador Tim Peters, descrevendo brevemente a filosofia do Python. Pode-se vê-lo através de um *easter egg* da linguagem pelo comando:
>>> ***import*** *this*

> The Zen of Python, by Tim Peters
>
> Beautiful is better than ugly.
> Explicit is better than implicit.
> Simple is better than complex.
> Complex is better than complicated.
> Flat is better than nested.
> Sparse is better than dense.
> Readability counts.
> Special cases aren't special enough to break the rules.
> Although practicality beats purity.
> Errors should never pass silently.
> Unless explicitly silenced.
> In the face of ambiguity, refuse the temptation to guess.
> There should be one– and preferably only one –obvious way to do it.
> Although that way may not be obvious at first unless you're Dutch.
> Now is better than never.
> Although never is often better than *right* now.
> If the implementation is hard to explain, it's a bad idea.
> If the implementation is easy to explain, it may be a good idea.
> Namespaces are one honking great idea – let's do more of those!

[1] CPython é a implementação principal da linguagem de programação Python, escrita em Linguagem C (mais especificamente o C89). É desenvolvida e mantida por Guido van Rossum e diversos outros desenvolvedores espalhados pelo mundo. O CPython é um interpretador de Bytecode. Ele possui uma interface funcional em diversas linguagens incluindo C, na qual os bindings podem ser escritos explicitamente em qualquer outra linguagem diferente de Python.Foundation (2021c, p. 40)

2 HISTÓRICO

A Linguagem Python foi concebida no fim dos anos 80, onde sua primeira ideia de implementação surgiu, mais especificamente em 1982, enquanto Guido Van Rossum trabalhava no CWI[1] em Amsterdã, Holanda, no time de desenvolvimento da Linguagem ABC. Posteriormente, em 1987, com o fim da linguagem ABC, Guido foi transferido para o grupo de trabalho Amoeba — um sistema operacional Microkernel liderado por Andrew Tanenbaum. Foi neste grupo que Guido percebeu a necessidade de uma linguagem para escrever programas intermediários, algo entre o C e o Shell Script. Tendo como base um código de demonstração da linguagem ABC, alguns elementos de sintaxe e a indentação obrigatória se tornaram grande fonte de inspiração para a nova linguagem.

Em 1989 o desenvolvimento do Python realmente teve início. Nos primeiros meses de 1990 o autor já possuía uma versão mínima e operacional e, pelo fim do ano de 1990, Python já era mais utilizada no CWI que a própria linguagem ABC.

No ano de 1991 Guido foi transferido do grupo Amoeba para o grupo Multimídia. De acordo com o próprio Guido em uma entrevista com Bill Vennes:

> "ABC me deu a inspiração crucial para Python, o grupo Amoeba a motivação imediata e o grupo de multimídia fomentou seu crescimento".(VENNERS, 2003)

Ainda neste ano, no dia 20 de Fevereiro, foi lançada a primeira versão do Python, então denominada de *v0.9.0* e anunciada no grupo de discussão alt.sources (*newsgroup*)[2]. A primeira release era composta de 21 partes que juntos formavam um arquivo *.tar*. Nesta primeira versão, o Python já contava com classes, herança, tratamento de exceções, funções, sistema de módulos (empresado da linguagem Modula-3) e os tipos de dado nativos *list, dict, str, e etc.*

Desde à primeira versão — e todas as outras versões lançadas dentro do CWI (Python 1.2) — possuíam uma licença derivada da licença MIT. Abaixo um pequeno histórico de todas as versões lançadas no CWI: [3]

É importante ressaltar que, apesar de a linguagem Python ter sido desenvolvida nas premissas do CWI, este não financiou ou providenciou fundos oficialmente para o desenvolvimento da linguagem.

---

[1] O Centrum Wiskunde & Informatica (Centro de Matemática e Ciência da Computação) é um centro de pesquisa na área de matemática e ciência da computação teórica que faz parte da Organização Holandesa de Pesquisa Científica (NWO). Está localizada no Amsterdam Science Park e é conhecido como o local de nascimento de várias linguagens como Algol, Algol 68, NetHack e XHTML. Foi membro fundador do Consórcio Europeu de Investigação para Informática e Matemática (ERCIM).

[2] O alt.sources é um *newsgroup* cujo propósito é ser um repositório de códigos fonte para pessoas que desejam distribuir e compartilhá-los para outras pessoas. Não há restrições do tipo, linguagem, máquina ou propósito para o código fonte.Kamens (2008)

[3] A versão 0.9.5 foi disponibilizada apenas para Machintosh

Tabela 1 – Histórico de versão.

Mês/Ano	versão
Fevereiro de 1991	0.9.0
Fevereiro de 1991	0.9.1
Outubro de 1991	0.9.2
Dezembro de 1991	0.9.4
Janeiro de 1992	0.9.5
Abril de 1992	0.9.6
Janeiro de 1993	0.9.8
Julho de 1993	0.9.9
Janeiro de 1994	1.0.0
Fevereiro de 1994	1.0.2
Maio de 1994	1.0.3
Julho de 1994	1.0.4
Outubro de 1994	1.1
Novembro de 1994	1.1.1
Abril de 1995	1.2

Fonte: Foundation (2021b)

2.1 NOME

No início de seu projeto, Guido não queria siglas ou um nome fraco, como era o caso da linguagem ABC, ele queria que o nome da linguagem fosse marcante e forte, mas não fazia questão que o nome possuísse um significado profundo. Guido então decidiu homenagear o grupo britânico de comédia: Monty Python's Flying Circus, o que se encaixou perfeitamente no "padrão" de nomear uma linguagem em homenagem a pessoas famosas[4] e à tradição do CWI de utilizar nomes de programas de TVs para projetos.

Por anos o autor evitou vincular a linguagem ao réptil (a cobra píton) mas desistiu quando a editora O'Reilly — que possui a tradição de utilizar animais nas capas de seus livros — sugeriu colocar uma cobra píton na capa do seu primeiro livro "Programming Python".

2.2 COMUNIDADE

A primeira "comunidade" do Python surgiu formalmente com a criação do *newsgroup: comp.lang.python na Usenet*, em março de 1993. Posteriormente, este *newsgroup* foi migrado para uma lista de discussão por e-mail, tendo como base o *GNU Mailman*, um gerenciador de listas software livre escrito em Python.

No verão de 1994, o grupo iniciou uma discussão intitulada "Se Guido fosse atingido por um ônibus?"[5]. Por mais mórbido que essa discussão soava, ela tocava no âmago da

[4] Vide: Pascal, Ada, Eiffel.
[5] McLay (1994, p. 1)

Figura 1 – Mount Python's Logo

Fonte: Wikipedia (2021)

comunidade Python, pois Guido era seu principal desenvolvedor e ele tomava as decisões, criando assim o medo do Python desaparecer com seu criador. Muitos justificavam que a política de "um homem só" reduziam as possibilidades de doação e investimento na linguagem. Visto isso, nesta discussão, nasceu a necessidade de se criar um padrão ou organização responsável pelo Python, desvinculando Guido como o único responsável (e detentor de seus direitos) e garantindo assim a existência prolongada da linguagem.

No início de 2000, Guido, Barry Warsaw, Jeremy Hylton e Fred Drake receberam o convite para ser juntar à *startup* BeOpen.com, uma iniciativa que estava recrutando diversos desenvolvedores *Open Source*. Antes de deixar a CNRI[6] os desenvolvedores foram forçados a lançar a versão 1.6, para finalizar o ciclo de desenvolvimento do Python.

Para a versão 1.6 a CNRI insistiu em utilizar uma licença escrita pelos seus próprios advogados. Como esperado, esta licença diferia da utilizada até o momento, visando controlar "os direitos do Python" e submetendo o software às leis do estado da Virginia. Como o Python era utilizado pelo GNU Mailman, a FSF (Free Software Foundation) estava receosa que essa nova licença pudesse restringir o uso de ambos os softwares. Desta forma, Richard Stallman e Eben Moglen (ambos da PSF[7]), analisaram a licença e chegaram a conclusão de que esta não era uma licença compatível com as premissas do software livre. Com o apoio de Eric Raymond e da PSF a licença foi reescrita

[6] A Corporation for National Research Initiatives (CNRI) é uma organização sem fins lucrativos formada em 1986 localizada em Virginia, Estados Unidos. Com o objetivo de empreender, fomentar e promover pesquisas de interesse público, as atividades giram em torno do desenvolvimento estratégico de tecnologias de informação baseadas em rede. A instituição fornece liderança e financiamento para pesquisa e desenvolvimento de infraestrutura de informação.CNRI (2020)

[7] Vide: página 6, secção: *Python Software Foundation*

para satisfazer tanto a FSF quanto a CNRI. A versão 1.6 foi lançada em Setembro de 2000, sendo que o grupo de desenvolvedores já estavam na BeOpen.com desde Maio de 2000.

Figura 2 – Free Software Foundation's Logo

Fonte: Foundation (2021a)

Devido a esta história do Python, a licença do Python era vista "em camadas". Na base tínhamos a licença do CWI, seguida pela licença do CNRI (no meio) e por último a licença da BeOpen.com. Apesar da confusão, a licença era compatível com o modelo OSI que define uma licença Open Source e também é compatível com a GNU GPL (General Public License), garantindo as liberdades de um software livre.

2.3 *BeOpen.com & Digital Creations*

Ja na BeOpen.com foi formado o grupo PythonLabs e a versão 2.0 do Python foi lançada em outubro de 2000. O Python 2.0[8] utilizava uma versão alterada da licença presente na versão 1.6 (alterando apenas o responsável para BeOpen.com). Nesta estadia o Python (como comunidade e linguagem) evoluiu significativamente:

- Os desenvolvedores passaram a se focar exclusivamente para o Python
- O desenvolvimento foi centralizado, utilizando um servidor CVS no SourceForge
- Por volta de 30 pessoas possuíam acesso de commit
- Banco de dados de patches e bugs também eram hospedados no SourceForge
- Criação das PEPs (Python Enhancement Proposal)

[8] A estadia na BeOpen.com rendeu apenas uma release do Python, a versão 2.0 citada anteriormente, pois em Outubro de 2000 ocorreu a falência e desmembramento da BeOpen.com e o PythonLabs foi contratado pela empresa Digital Creations.

Figura 3 – Zope Logo

Fonte: Project e Community (2021)

Em paralelo à esta contratação [*Digital Creations*], o PythonLabs recebeu também convites de outras duas empresas, a VA Linux e a ActiveState. Posteriormente a Digital Criations mudou de nome e ficou conhecida como Zope Corporation, referência ao seu produto mais conhecido, o Web CMS (Content Managing System) Zope. Parte da mudança para da Zope Corporaton foi influenciada pela certeza de que o futuro do Python não podia ser influenciado pelos objetivos e ideais daqueles para os quais Guido trabalhava. Foi então que criaram a Python Software Foundation (PSF).

2.4 *Python Software Foundation*

Python Software Foundation Em 2001 foi criada a Python Software Foundation (PSF), uma organização sem fins lucrativos constituída por membros da equipe de desenvolvimento (daquela época) e por Eric Raymond. Ela tem como objetivo ser dona de qualquer propriedade intelectual relacionada ao Python, e como missão promover e proteger o avanço da linguagem Python, além suportar e auxiliar o crescimento de comunidades de programadores Python. Ela possui diversos patrocinadores como:

- ActiveState;
- Advanced Simulation Technology Inc. (ASTi);
- Array BioPharma, Inc.;
- BizRate.com;
- Canonical;
- Globo;
- Google;

- Lucasfilm;
- Microsoft;
- OpenEye Scientific Software;
- O'Reilly Media, Inc.;
- Red Hat;

Em dezembro de 2008 é lançada a versão 3 do Python, mas ainda manteve muitos adeptos na versão 2, tanto que ambas são retrocompatíveis apesar da 2ª não ser mais recomendada para novos projetos.

Após a criação da PSF todas as *releases* desde a 2.1 foram feitas utilizando a PSF License Agreement, uma licença que atribui todos os direitos do Python à PSF. A licença está disponível na íntegra na documentação oficial do Python. Uma vez que o futuro do Python (e a sua evolução) se desvinculou dos empregadores de seu criado, existem poucos relatos e registros. Segue alguns destaques:

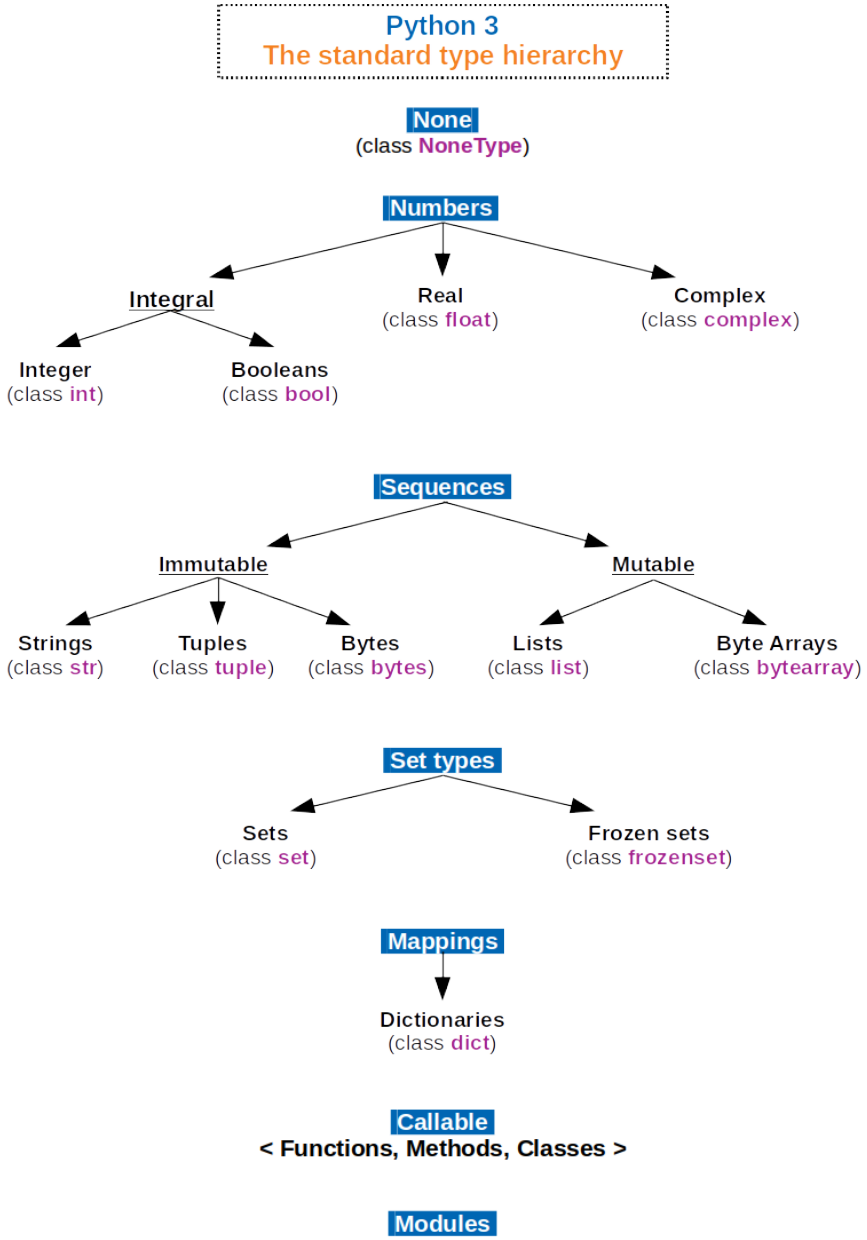
- Em Julho de 2003 o PythonLab saiu da Zope Corporation para trabalhar na Elemental Security em San Mateo, California;
- Em Dezembro de 2005 Guido foi trabalhar no Google em Mountain View, Califórnia;
- Em Janeiro de 2013 Guido foi trabalhar para o Dropbox.

3 PARADIGMA

3.1 CONSTRUÇÕES

Construções de Python incluem: estrutura de seleção *(if, else, elif)*; estrutura de repetição *(for, while)*, que itera por um container, capturando cada elemento em uma variável local dada; construção de classes (class); construção de sub-rotinas *(def)*; construção de escopo *(with)*, como por exemplo para adquirir um recurso.

Figura 4 – Python 3 The Standard type hyerarchy



Fonte: <https://pt.wikipedia.org/wiki/Ficheiro:Python_3._The_standard_type_hierarchy.png>

3.2 TIPOS DE DADOS

A tipagem de Python é forte, pois os valores e objetos têm tipos bem definidos e não sofrem coerções como em C ou Perl. São disponibilizados diversos tipos de dados nativos:[1] [2]

Tabela 2 – Tipos de dados em Python.

Tipo de dados	Descrição	Exemplo de sintaxe
str, unicode	Uma cadeia de caracteres imutável	'foo'; "bar"
list	Lista heterogênea mutável	[4.0, 'string', True]
tuple	Tupla imutável	(4.0, 'string', True)
set, frozenset	Conjunto não ordenado	set([4.0, 'string', True])
dict	conjunto associativo	{'key1': 1.0, 'key2': False}
int	Número de precisão fixa	42; 2147483648L
float	Ponto flutuante	3.1415927
complex	Número complexo	$3+2j$
bool	Booleano	True; False
!=	Diferente	Diferente

Fonte: Foundation (2021b)

Python também permite a definição dos tipos de dados próprios, através de classes. Instâncias são construídas invocando a classe *(FooClass())*, e as classes são instância da classe type, o que permite metaprogramação e reflexão. Métodos são definidos como funções anexadas à classe, e a sintaxe *instância.método(argumento)* é um atalho para *Classe.método(instância, argumento)*. Os métodos devem referenciar explicitamente a referência para o objeto incluindo o parâmetro self como o primeiro argumento do método. Antes da versão 3.0, Python possuía dois tipos de classes: "old-style"e "new-style". Classes old-style foram eliminadas no Python 3.0, e todas são new-style. Em versões entre 2.2 e 3.0, ambos tipos de classes podiam ser usadas. A sintaxe de ambos estilos é a mesma, a diferença acaba sendo de onde objeto da classe é herdado, direta ou indiretamente (todas classes new-style herdam de objeto e são instâncias de type). As classes new-styles nada mais são que tipos definidos pelo usuário.

3.3 PALAVRAS RESERVADAS

O Python 3 define as seguintes palavras reservadas:

- **False**
- **None**
- **True**

[1] O tipo int (inteiro) é transparentemente convertido para long caso não caiba em um int.
[2] Os Tipos set e frozenset não contém elementos duplicados

- **and**
- **as**
- **assert**
- **break**
- **class**
- **continue**
- **def**
- **del**
- **elif**
- **else**
- **except**
- **finally**
- **for**
- **from**
- **global**
- **if**
- **import**
- **in**
- **is**
- **lambda**
- **not**
- **nonlocal**
- **or**
- **pass**
- **raise**
- **try**
- **return**
- **while**
- **with**
- **yield**

3.4 OPERADORES

Os operadores básicos de comparação como ==, <, >=, entre outros são usados em todos os tipos de dados, como números, cadeias de texto, listas e mapeamentos. Comparações em cadeia como $a < b < c$ possuem o mesmo significado básico que na matemática: os termos são comparadas na ordem. É garantido que o processamento da expressão lógica irá terminar tão cedo o veredito seja claro, o princípio da avaliação mínima. Usando a expressão anterior, se $a < b$ é falso, c não é avaliado.

Quanto aos operadores lógicos, até Python 2.2 não havia o tipo de dado booleano.

Em todas as versões da linguagem os operadores lógicos[3] tratam *, 0, None, 0.0, [] e* como falso, enquanto o restante é tratado como verdadeiro de modo geral. A comparação binária retorna uma das duas constantes acima.

Os operadores booleanos and e or também seguem a avaliação mínima. Por exemplo, $y == 0 or x/y > 100$ nunca lançará a exceção de divisão por zero.

3.5 INTERPRETADOR INTERATIVO

O interpretador interativo é uma característica diferencial da linguagem, porque há a possibilidade de testar o código de um programa e receber o resultado em tempo real, antes de iniciar a compilação ou incluí-las nos programas. Por exemplo:[4]

Algoritmo 1: Exemplo do Interpretador Interativo

```
>>> 1+1
2
>>>
>>> a = 1+1
>>> print a
2
>>> print(a)
2
>>>
```

[3]Na versão 2.2.1 as constantes True e False foram adicionadas (subclasses de 1 e 0 respectivamente)

[4]A partir da versão 3.0, o comando print passou a ser uma função, sendo obrigatório o uso de parênteses.

4 CARACTERÍSTICAS MARCANTES

4.1 ANÁLISE LÉXICA

De acordo com Rossum e Drake (2011), a análise léxica é uma análise do interpretador, os programas são lidos por um analisador sintático que divide o código em *tokens*. Todo programa é dividido em linhas lógicas que são separadas pelo *token NEWLINE*, as linhas físicas são trechos de código divididos pelo caractere *ENTER*. Linhas lógicas não podem ultrapassar linhas físicas com exceção de junção de linhas, por exemplo:

Algoritmo 2: Exemplo do Interpretador Interativo

```
if resultado > 2 and 1 <= 5 and 2 < 5:
print('Resultado: %f' % d)
ou
MESES COMO = ['janeiro','fevereiro','maro',
'abril','maio','junho',
'julho','agosto','setembro',
'outubro','novembro','dezembro']
```

Para a delimitação de blocos de códigos, os delimitadores são colocados em uma pilha e diferenciados por sua indentação, iniciando a pilha com valor 0 (zero) e colocando valores maiores que os anteriores. Para cada começo de linha, o nível de indentação é comparado com o valor do topo da pilha.

- Se o número da linha for igual ao topo da pilha, a pilha não é alterada.
- Se o valor for maior, a pilha recebe o nível de indentação da linha e o nome *INDENT (psuh)*.
- Se o nível de indentação for menor, então é desempilhado até chegar a um nível de indentação recebendo o nome *DEDENT (pop)*.
- Se não encontrar nenhum valor, é gerado um erro de indentação.

Abaixo um exemplo de permutação, retirado do capítulo 2.1 sobre Estrutura de linhas na Análise léxica do Manual de Referência da linguagem[1]:

Python foi desenvolvido para ser uma linguagem de fácil leitura, com um visual agradável, frequentemente usando palavras e não pontuações como em outras linguagens. Para a separação de blocos de código, a linguagem usa espaços em branco e indentação ao invés de delimitadores visuais como chaves (C, Java) ou palavras (BASIC, Fortran, Pascal). Diferente de linguagens com delimitadores visuais de blocos, em Python a indentação é obrigatória. O aumento da indentação indica o início de um novo bloco, que termina da diminuição da indentação.

[1] Rossum e Drake (2011, p. 37), vide figura 5: p. 13, seção: ANÁLISE LÉXICA

Figura 5 – Indentação em Python

```
        def perm(l):                    NOVA LINHA
INDENT      if len(l) <= 1:             NOVA LINHA
INDENT        return[1]                 NOVA LINHA
DEDENT      r = [ ]                     NOVA LINHA
            for i in range(len(l)):     NOVA LINHA
INDENT        s = l[:i] + l[i+1:]       NOVA LINHA
              p = perm(s)               NOVA LINHA
DEDENT      for x in p:                 NOVA LINHA
INDENT        r.append(l[i:i+1]+x)      NOVA LINHA
DEDENT      return r
```

Fonte: Rossum e Drake (2011)

Usando um editor de texto comum é muito fácil existirem erros de indentação, o recomendado é configurar o editor conforme a análise léxica do Python ou utilizar uma IDE. Todas as IDE que suportam a linguagem fazem indentação automaticamente. Exemplo:

Figura 6 – Indentação em Python

Indentação correta

```
def valor1():
    while True:
        try:
            c =
int(input('Primeiro Valor: '))
            return c
        except ValueError:
            print 'Inválido!'
```

Indentação incorreta

```
def valor1():
while True:
try:
c = int(input('Primeiro Valor: '))
return c
except ValueError:
print 'Inválido!'
```

Fonte: Rossum e Drake (2011)

O analisador léxico reconhecerá as palavras reservadas while, def, try, except, return, print e as cadeias de caracteres entre aspas simples e a indentação, e se não houver problemas o programa executará normalmente, senão apresentará a exceção: *IndentationError*[2].

4.2 COMPILADOR DE BYTECODE

A linguagem é de altíssimo nível, como já dito, mas ela também pode compilar seus programas para que a próxima vez que o executar não precise compilar o novamente,

[2] Foundation (2021c, p. 20)

reduzindo o tempo de carga na execução.

Utilizando o interpretador interativo não é necessário a criação do arquivo de Python compilado, os comandos são executados interativamente. Entretanto, quando um programa ou um módulo é evocado, o interpretador realiza a análise léxica e sintática, compila o código de alto nível se necessário e o executa na máquina virtual da linguagem.

O bytecode é armazenado em arquivos com extensão *.pyc ou .pyo*, este último no caso de bytecode otimizado[3].

Normalmente, o Python trabalha com dois grupos de arquivos:

- Os módulos do núcleo da linguagem, sua biblioteca padrão e os módulos independentes, criados pelo usuário.
- No núcleo do interpretador existe o analisador léxico, o analisador sintático que utiliza Estruturas de Objetos (tempo de execução), o Compilador que aloca memória (tempo de execução) e depois do Avaliador de código que modifica o estado atual do programa (tempo de execução), mostrando resultado para o usuário.

4.3 ORIENTAÇÃO A OBJETO

Python suporta a maioria das técnicas da programação orientada a objeto. Qualquer objeto pode ser usado para qualquer tipo, e o código funcionará enquanto haja métodos e atributos adequados. O conceito de objeto na linguagem é bastante abrangente: classes, funções, números e módulos são todos considerados objetos. Também há suporte para metaclasses, polimorfismo, e herança (inclusive herança múltipla). Há um suporte limitado para variáveis privadas.

Na versão 2.2 de Python foi introduzido um novo estilo de classes em que objetos e tipos foram unificados, permitindo a especialização de tipos. Já a partir da versão 2.3 foi introduzido um novo método de resolução de ambiguidades para heranças múltiplas.(SIMIONATO, 2008)

Uma classe é definida com *class* nome, e o código seguinte é a composição dos atributos. Todos os métodos da classe recebem uma referência a uma instância da própria classe como seu primeiro argumento, e a convenção é que se chame este argumento self. Assim os métodos são chamados objeto.método(argumento1, argumento2) e são definidos iguais a uma função, como método(self, argumento1, argumento2). Veja que o parâmetro self conterá uma referência para a instância da classe definida em objeto quando for efetuada esta chamada. Os atributos da classe podem ser acessados em qualquer lugar da classe, e os atributos de instância (ou variável de instância) devem ser declarados dentro dos métodos utilizando a referência à instância atual (self)

Em Python não existe proteção dos membros duma classe ou instância pelo interpretador, o chamado encapsulamento. Convenciona-se que atributos com o nome

[3]Interessante notar que o bytecode da linguagem também é de alto nível, ou seja, é mais legível aos seres humanos que o código de byte do C, por exemplo.

começando com um são de uso privado da classe, mas não há um policiamento do interpretador contra acesso a estes atributos. Uma exceção são nomes começando com, no caso em que o interpretador modifica o nome do atributo.

Python permite polimorfismo, que condiz com a reutilização de código. É fato que funções semelhantes em várias partes do software sejam utilizadas várias vezes, então definimos esta função como uma biblioteca e todas as outras funções que precisarem desta a chamam sem a necessidade de reescrevê-la.

Python não possui overloading; não é possível criar duas funções com o mesmo nome, pois elas são consideradas atributos da classe. Caso o nome da função se repita em outra assinatura, o interpretador considera esta última como override e sobrescreve a função anterior. Algumas operações entre diferentes tipos são realizadas através de coerção ($ex.: 3.2 + 3$).

É possível encapsular abstrações em módulos e pacotes. Quando um arquivo é criado com a extensão *.py*, ele automaticamente define um módulo. Um diretório com vários módulos é chamado de pacote e deve conter um modulo chamado *__init__*, para defini-lo como principal. Estas diferenciações ocorrem apenas no sistema de arquivos. Os objetos criados são sempre módulos. Caso o código não defina qual dos módulos será importado, o padrão é o *__init__*.

4.4 PROGRAMAÇÃO FUNCIONAL

Uma das construções funcionais de Python é a compreensão de listas[4], uma forma eficiente de construir listas. Por exemplo, pode-se usar a técnica para calcular as cinco primeiras potências de dois. O algoritmo quicksort também pode ser expresso usando a mesma técnica.

Em Python, funções são objetos de primeira classe que podem ser criados e armazenados dinamicamente. O suporte a funções anônimas está na construção lambda (Cálculo-λ). Não há disponibilidade de funções anônimas de fato, pois os lambdas contêm somente expressões e não blocos de código.

Python também suporta clausuras léxicas desde a versão 2.2. Já geradores foram introduzidos na versão 2.2 e finalizados na versão 2.3, e representam o mecanismo de Python para a avaliação preguiçosa[5] de funções.

[4]Compreensão de lista é uma construção sintática disponível em algumas linguagens de programação para criação de uma lista baseada em listas existentes. Ela segue a forma da notação de definição de conjunto matemática (compreensão de conjunto) como forma distinta para uso de funções de mapa e filtro. Howe (2005)

[5]Avaliação preguiçosa (também conhecida por avaliação atrasada) é uma técnica usada em programação para atrasar a computação até um ponto em que o resultado da computação é considerado suficiente, o necessário. Os benefícios da avaliação preguiçosa incluem o aumento do desempenho ao evitar cálculos desnecessários, evitando condições de erro na avaliação de expressões compostas, a habilidade em construir estruturas de dados infinitas e a habilidade de definir estruturas do controle como funções regulares melhor que usando primitivas internas. No oposto de avaliação atrasada está avaliação ansiosa, também conhecido como avaliação rigorosa. Watt (2004)

4.5 TRATAMENTO DE EXCEÇÕES

Python suporta e faz uso constante de tratamento de exceções como uma forma de testar condições de erro e outros eventos inesperados no programa. É inclusive possível capturar uma exceção causada por um erro de sintaxe. O estilo da linguagem apoia o uso de exceções sempre que uma condição de erro pode aparecer. Por exemplo, ao invés de testar a disponibilidade de acesso a um recurso, a convenção é simplesmente tentar usar o recurso e capturar a exceção caso o acesso seja rejeitado.

Exceções são usadas frequentemente como uma estrutura de seleção, substituindo blocos *if-else*, especialmente em situações que envolvem threads. Uma convenção de codificação é o EAFP, do inglês, "é mais fácil pedir perdão que permissão". Isso significa que, em relação a desempenho, é preferível capturar exceções do que testar atributos antes de os usar. Segue abaixo exemplos de código que testam atributos e que capturam exceções:

Figura 7 – Mount Python's Logo

Teste de atributo

```
if hasattr(spam, 'eggs'):
    ham = spam.eggs
else:
    handle_error()
```

Captura de exceção

```
try:
    ham = spam.eggs
except AttributeError:
    handle_error()
```

Fonte: Rossum e Drake (2011)

Ambos os códigos produzem o mesmo efeito, mas há diferenças de desempenho. Quando spam possui o atributo eggs, o código que captura exceções é mais rápido. Caso contrário, a captura da exceção representa uma perda considerável de desempenho, e o código que testa o atributo é mais rápido. Geralmente, o paradigma da captura de exceções é mais rápido, e também pode evitar problemas de concorrência. Por exemplo, num ambiente multitarefa, o espaço de tempo entre o teste do atributo e seu uso de fato pode invalidar o atributo, problema que não acontece no caso da captura de exceções.

4.6 BIBLIOTECA PADRÃO

Python possui uma grande biblioteca padrão[6], geralmente citada como um dos maiores trunfos da linguagem, fornecendo ferramentas para diversas tarefas. Por conta da grande variedade de ferramentas fornecida pela biblioteca padrão, combinada com a habilidade de usar linguagens de nível mais baixo como C e C++, Python pode ser poderosa para conectar componentes diversos de software.

[6] Algumas partes da biblioteca são cobertas por especificações (por exemplo, a implementação WSGI da wsgiref segue o PEP 333), mas a maioria dos módulos não o segue.

A biblioteca padrão conta com facilidades para escrever aplicações para a Internet, contando com diversos formatos e protocolos como MIME e HTTP. Também há módulos para criar interfaces gráficas, conectar em bancos de dados relacionais e manipular expressões regulares.

4.7 INTEROPERABILIDADE

Outro ponto forte da linguagem é sua capacidade de interoperar com várias outras linguagens, principalmente código nativo. A documentação da linguagem inclui exemplos de como usar a Python C-API para escrever funções em C[7] que podem ser chamadas diretamente de código Python. A biblioteca Boost do C++ inclui uma biblioteca para permitir a interoperabilidade entre às duas linguagens, e pacotes científicos usam bibliotecas de alto desempenho numérico escritos em Fortran e mantidos há décadas.

4.8 COMENTÁRIOS

Python fornece duas alternativas para documentar o código. A primeira é o uso de comentários para indicar o que certo código faz. Comentários começam com # e são terminados pela quebra da linha. Não há suporte para comentários que se estendem por mais de uma linha; cada linha consecutiva de comentário deve indicar #. A segunda alternativa é o uso de cadeias de caractere, literais de texto inseridos no código sem atribuição. Cadeias de caracteres em Python são delimitadas por ” ou ′ para única linha e por ” ou ′′′ para múltiplas linhas. Entretanto, é convenção usar os métodos de múltiplas linhas em ambos os casos.

Diferente de comentários, as cadeias de caracteres usadas como documentação são objetos Python e fazem parte do código interpretado. Isso significa que um programa pode acessar sua própria documentação e manipular a informação. Há ferramentas que extraem automaticamente essa documentação para a geração da documentação de API a partir do código. Documentação através de cadeias de caracteres também pode ser acessada a partir do interpretador através da função *help()*.

[7] Mas atualmente esse *sequer* é o modo mais indicado de interoperação, havendo alternativas tais como Cython, Swig ou cffi

5 LINGUAGENS SIMILARES OU CONFLITANTES

Python é frequentemente comparado a outras linguagens interpretadas, como Java, JavaScript, Perl, Tcl, Smalltalk, C++, Common Lisp e Scheme(FILIPA, 2020). Na prática, a escolha de uma linguagem de programação é frequentemente ditada por outras restrições reais, como custo, disponibilidade, treinamento e investimento inicial e até mesmo compromisso emocional.

O subconjunto “baseado em objeto” do Python é equivalente ao JavaScript e, como o JS, o Python suporta um estilo de programação que usa funções e variáveis simples sem se envolver em definições de classe. No entanto, para JavaScript, isso é tudo o que existe.

Espera-se que os programas Python sejam mais lentos do que os programas Java, porém levam menos tempo para serem desenvolvidos. Os programas Python são tipicamente 3 a 5 vezes mais baixos do que programas Java comparáveis. Essa diferença pode ser atribuída aos tipos de dados de alto nível integrados do Python e sua tipagem dinâmica. Devido à digitação em tempo de execução, o tempo de execução do Python terá um desempenho melhor do que o Java. Por exemplo, ao avaliar a expressão a + b, você deve primeiro examinar os objetos AEB para descobrir seu tipo, que é desconhecido em tempo de compilação. Por essas razões, Python é muito mais adequado como uma linguagem ”cola”, enquanto Java é melhor caracterizado como uma linguagem de implementação de baixo nível. Na verdade, os dois juntos formam uma ótima combinação. Os componentes podem ser desenvolvidos em Java e combinados para formar aplicativos Python; Python também pode ser usado para prototipar componentes até que seu design possa ser ”reforçado”em uma implementação Java.

Python brilha como uma linguagem de cola, usada para combinar componentes escritos em C++. Como comparado em java, um programa tem de 3 a 5 vezes maior [nível], vemos essa diferença aumentar em relação de 5 a 10 vezes numa aplicação de C++ compara Python.

Por sua vez, o Tcl, que tradicionalmente armazena todos os dados como strings, é fraco em estruturas de dados e o código regular é executado muito mais devagar do que o Python. Portanto, enquanto um grande aplicativo Tcl ”típico”geralmente contém extensões Tcl escritas em C ou C ++ que são específicas para esse aplicativo, aplicativos Python equivalentes podem ser escritos em ”Python puro”. Obviamente, o desenvolvimento puro em Python é muito mais rápido do que escrever e depurar um componente C ou C ++. Python adotou a interface Tk como sua biblioteca padrão de componentes GUI.

Em Smalltalk, talvez a maior diferença entre eles seja a sintaxe ”convencional”do Python, que oferece uma vantagem no treinamento de programadores. No entanto, ele distingue os tipos de objeto embutidos das classes definidas pelo usuário e atualmente não permite a herança de tipos embutidos. Python tem uma filosofia diferente em relação ao

ambiente de desenvolvimento e distribuição de código. Enquanto Smalltalk tradicionalmente tem uma ”imagem de sistema”monolítica, que inclui o ambiente e o programa do usuário, o Python armazena módulos padrão e módulos do usuário em arquivos individuais que podem ser facilmente reorganizados ou distribuídos fora do sistema.

Python e Perl vêm de um plano de fundo semelhante e possuem muitas características semelhantes, mas têm uma filosofia diferente. Python enfatiza o suporte para metodologias de programação comuns, como desenho de estrutura de dados e programação orientada a objetos, e encoraja os programadores a escrever um código legível , fornecendo uma notação elegante, mas não excessivamente encriptada.

Em relação a common e lisp e Scheme, essas linguagens eram tão próximas dele em sua semântica dinâmica, mas tão diferentes em suas abordagens sintáticas que a comparação se tornou quase um argumento religioso: a falta de sintaxe de Lisp é vantagem ou desvantagem? Deve-se notar que o Python possui recursos de introspecção semelhantes aos do Lisp e que os programas Python podem construir e executar fragmentos de programa em tempo real. Freqüentemente, as propriedades do mundo real são decisivas: Common Lisp é grande e o mundo Scheme é fragmentado entre muitas versões incompatíveis, da qual Python tem uma implementação única, gratuita e compacta.

6 CONCLUSÃO

6.1 CONSIDERAÇÕES FINAIS

Python é uma linguagem fascinante. De acordo com o site Stack Overflow[1], python é a linguagem mais amada e mais utilizada atualmente. Com vários adeptos e uma rica comunidade, a linguagem continua em seu crescimento. Ela é versátil em fazer programas rápidos, *Data Science*, segurança da informação e diversas outras áreas, se tornando um "canivete suíço" da programação. Apesar da sua popularidade recente (a partir do Python 2), ela era extensivamente utilizada em uma série de aplicações desde a década de 90.

Atualmente, Python começou a ser utilizado como linguagem introdutória por várias instituições de ensino sendo atualmente a mais recomendada para iniciantes, apesar do conhecimento de baixo nível não seja profundo. Certamente, a linguagem possui um futuro promissor pela frente na inovação da área científica, educacional, entre outras.

[1] O Stack Overflow survey é a maior pesquisa sobre programadores do mundo, no ano de 2020. "Este ano [2020], em vez de almejar ser o maior, nos propusemos a tornar nossa pesquisa mais representativa da diversidade de programadores em todo o mundo. Dito isso, a pesquisa ainda é grande. A pesquisa deste ano foi realizada por quase 65.000 pessoas."Overflow (2020). Outra pesquisa relevante que também aponta resultados similares é a Interactive: The Top Programming Languages da IEEE, onde Python também é a mais usada Spectrum (2020).

Referências

CNRI. **About CNRI**. 2020. Disponível em: <https://www.cnri.reston.va.us/about_cnri.html>. Acesso em: 1 de junho de 2021. Citado na página 4.

FILIPA, S. **Python: Comparando Python com outras linguagens**. 2020. Disponível em: <https://www.cetax.com.br/blog/comparando-python-com-outras-linguagens/>. Acesso em: 1 de junho de 2021. Citado na página 18.

FOUNDATION, F. S. **Free Software Foundation**. 2021. Disponível em: <https://www.fsf.org/about/>. Acesso em: 31 de maio de 2021. Citado na página 5.

FOUNDATION, P. S. **Python Official Documentation**. 2021. Disponível em: <https://docs.python.org/3/>. Acesso em: 31 de maio de 2021. Citado 2 vezes nas páginas 3 e 9.

FOUNDATION, P. S. **Python/C API Reference Manua**: Initialization, finalization, and threads. [S.l.], 2021. 90 p. Disponível em: <https://docs.python.org/3/c-api/index.html>. Acesso em: 01 de fevereiro de 2021. Citado 2 vezes nas páginas 1 e 13.

HOWE, D. **List Comprehension**. 2005. Disponível em: <https://web.archive.org/web/20050125080818/http://ftp.sunet.se/foldoc/foldoc.cgi?list+comprehension>. Acesso em: 1 de junho de 2021. Citado na página 15.

KAMENS, J. I. **alt-sources-intro**. 2008. Disponível em: <http://www.faqs.org/faqs/alt-sources-intro/>. Acesso em: 1 de junho de 2021. Citado na página 2.

MCLAY, M. **If Guido was hit by a bus?** 1994. Disponível em: <https://legacy.python.org/search/hypermail/python-1994q2/1040.html>. Acesso em: 1 de junho de 2021. Citado na página 3.

OVERFLOW, S. **2020 Developer Survey**. 2020. Disponível em: <https://insights.stackoverflow.com/survey/2020#overview>. Acesso em: 1 de junho de 2021. Citado na página 20.

PROJECT, Z.; COMMUNITY. **Zope Official Documentation**. 2021. Disponível em: <https://www.zope.org/documentation/index.html>. Acesso em: 31 de maio de 2021. Citado na página 6.

ROSSUM, G. van; DRAKE, F. L. **The Python Language Reference Manual**. [S.l.]: Network Theory Ltd., 2011. ISBN 1906966141. Citado 3 vezes nas páginas 12, 13 e 16.

SIMIONATO, M. **The Python 2.3 Method Resolution Order**: Documentos técnicos e científicos brasileiros compatíveis com as normas abnt. [S.l.], 2008. 20 p. Disponível em: <https://www.python.org/download/releases/2.3/mro/>. Acesso em: 31 de maio de 2021. Citado na página 14.

SPECTRUM, I. **Interactive: The Top Programming Languages**. 2020. Disponível em: <https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2020>. Acesso em: 1 de junho de 2021. Citado na página 20.

VENNERS, B. **The Making of Python: A Conversation with Guido van Rossum, Part I**. 2003. Disponível em: <https://www.artima.com/articles/the-making-of-python>. Acesso em: 1 de junho de 2021. Citado na página 2.

WATT, D. A. **Programming Language Design Concepts**. 1. ed. University of Glasgow: John Wiley & Sons, 2004. Citado na página 15.

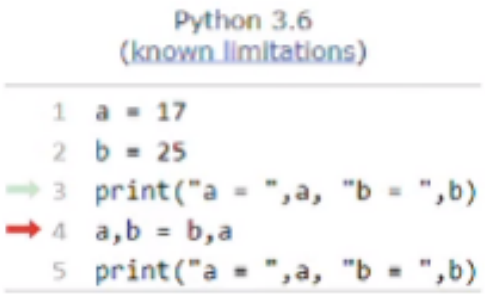
WIKIPEDIA. **Mount Python's Flying Circus**. 2021. Disponível em: <https://en.wikipedia.org/wiki/Monty_Python>. Acesso em: 31 de maio de 2021. Citado na página 4.

Apêndices

APÊNDICE A – Opinião: Guilherme Cosso

O que mais me chamou a atenção na linguagem tema Python foi a sua nomenclatura baseada no seriado Monty Python. O seu criador Guildo Van Russo queria uma linguagem forte para a LP. Apesar de todos associarem a Lp as cobras pithón. As trocas de variáveis sem conter uma terceira para fazer a troca, além de não ser necessário ponto e virgula ";", chaves "{"e a Main como pode ser visto abaixo:

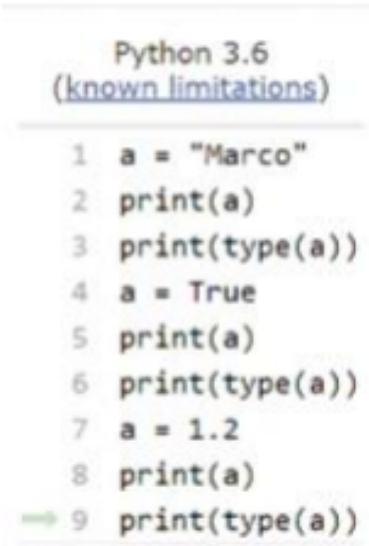
Figura 8 – Exemplo de Código



Fonte: Original

A tipagem fraca e o seu conteúdo ser dinamicamente tipada: o tipo de conteúdo de variável poder mudar automaticamente.

Figura 9 – Exemplo de Tipagem



Fonte: Original

APÊNDICE B – Opinião: Iyan Lucas

Recentemente, comecei a exercer minha iniciação científica sob o Observatório da Saúde. Comecei a gostar de séries temporais e, apesar do meu apreço por linguagens como Java, C# e C++, Python era essencial para tudo, desde a análise de dados à plotar pontos em mapas. Fui, entre muitas áspas, ”forçado”ao aprendizado da linguagem. Percebi de imediato, a minha estranheza com a falta de ponto-e-vírgula e chaves, sobretudo a ausência de uma main.

Agora, ao redigir este apêndice, estou bem acostumado ao Python, programando todo dia, todo meses com a linguagem e então me realizei da grandeza da LP. Ela é, apesar de confusa, funcional, curta, simples e prática. Fiquei assustado pela eficiência dela com grandes bases de dados e rapidez. Apesar de que eu penso que faria os programas mais organizados e mais intuitivamente em Java, percebi que essa visão minha é só um ponto de vista de muitos.

APÊNDICE C – Opinião: Samir Cambraia

Depois da pesquisa, minha visão sobre a linguagem ficou mais abrangente. Python é uma linguagem promissora com um grande currículo em aplicações em diversas áreas sendo a mais competente em em sua maioria, tendo sua simplicidade no aprendizado a programação possibilitando o conhecimento de rápida teoria/prática.

Porém, na minha opinião as estruturas e regras que levam a grande possibilidade de grandes códigos serem executados com menos trabalho e ”linhas”é que me incomoda pelo fato de, como em java, se ter um controle maior e não esperar que a linguagem em si possibilite sem saber o que ocorre por trás assim pessoas com novas experiências, na minha opinião, ficaria rasas em conhecimento como na parte de memória e outros conhecimentos conquistados em linguagens de baixo nível que possibilitam maior controle.