JAVA SCRIPT

JavaScript é uma linguagem de programação que permite implementar funcionalidades mais complexas em páginas web. Sempre que uma página web faz mais do que apenas mostrar informações estáticas para você - ela mostra em tempo real conteúdos atualizados, mapas interativos, animações gráficas em 2D/3D, vídeos, etc. - você pode apostar que o Javascript provavelmente está envolvido.

O caminho para o aprendizado

O JavaScript não é tão fácil de aprender como outras linguagens, como HTML e CSS, que são outros dois pilares do desenvolvimento front-end. Antes de tentar aprender JavaScript, é altamente recomendável que você aprenda e se familiarize com pelo menos estas duas tecnologias primeiro. Você pode começar através dos seguintes módulos:

Começando na Web

Introdução ao HTML

Introdução ao CSS

Possuir experiência em outras linguagens de programação pode também ser útil.

Depois de aprender o básico de JavaScript, você estará apto a estudar tópicos mais avançados, como:

JavaScript aprofundado, como ensinado em Guia JavaScript

Referências API Web

Módulos

Este tópico contém os seguintes módulos, em uma ordem que sugerimos para estudá-los.

Primeiros passos em JavaScript

Em nosso primeiro módulo JavaScript, primeiro responderemos algumas questões fundamentais como "o que é JavaScript?", "Como ele se parece?" E "o que ele pode fazer?", antes de passar para sua primeira experiência prática de escrever JavaScript. Depois disso, discutimos alguns recursos chave do JavaScript em detalhes, como variáveis, cadeias de caracteres, números e matrizes.

Blocos de codigo JavaScript

Neste módulo, continuaremos a falar sobre os principais recursos fundamentais do JavaScript, voltando nossa atenção para os tipos mais comuns de blocos de código, como instruções

condicionais, funções e eventos. Você já viu essas coisas no curso, mas apenas de passagem, aqui discutiremos tudo explicitamente.

Introdução a objetos em JavaScript

Em JavaScript, a maioria das coisas são objetos, desde seus principais recursos até as APIs do navegador. Você pode até criar seus próprios objetos.É importante entender a natureza orientada a objetos do JavaScript se você quiser ir mais longe com seu conhecimento da linguagem e escrever um código mais eficiente, portanto, fornecemos este módulo para ajudá-lo. Aqui ensinamos a teoria e a sintaxe de objetos em detalhes, observamos como criar seus próprios objetos e explicamos quais são os dados JSON e como trabalhar com eles.

JavaScript Assíncrono

Neste módulo, examinamos o JavaScript assíncrono, por que é importante e como ele pode ser usado para lidar efetivamente com possíveis operações de bloqueio, como a busca de recursos de um servidor.

API's Web do lado cliente

Ao escrever JavaScript para sites ou aplicativos da Web, você não vai muito longe antes de começar a usar APIs - interfaces para manipular diferentes aspectos do navegador e do sistema operacional em que o site está sendo executado, ou até dados de outros sites ou serviços. Neste módulo, vamos explorar o que são as APIs e como usar algumas das APIs mais comuns que você encontrará com frequência em seu trabalho de desenvolvimento.

Aprofundando

Uma vez que você se acostumar com o mundo do JavaScript aqui estão alguns outros módulos em que você pode mergulhar:

Biblioteca de referência JavaScript

Em nossa extensa biblioteca de referência você encontrará cada aspecto de Javascript tratado em detalhes: objetos globais, operadores, declarações e funções.

Introdução à Orientação a Objetos (OO) em JavaScript

Introdução aos conceitos de Programação orientada a objetos em JavaScript.

Resolvendo problemas comuns com Javascript

Use Javascript para resolver problemas comuns, este link proporciona contéudos explicativos de como usar o JavaScript para solucionar problemas muito comuns ao criar uma página da Web.

PYTHON

Python é uma linguagem de programação de alto nível,[5] interpretada, de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi lançada por Guido van Rossum em 1991.[1] 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.

A linguagem foi projetada com a filosofia de enfatizar a importância do esforço do programador sobre o esforço computacional. Prioriza a legibilidade do código sobre a velocidade ou expressividade. Combina uma sintaxe concisa e clara com os recursos poderosos de sua biblioteca padrão e por módulos e frameworks desenvolvidos por terceiros.

Python é uma linguagem de propósito geral de alto nível, multiparadigma, suporta o paradigma orientado a objetos, imperativo, funcional e procedural. Possui tipagem dinâmica e uma de suas principais características é permitir a fácil leitura do código e exigir poucas linhas de código se comparado ao mesmo programa em outras linguagens. Devido às suas características, ela é principalmente utilizada para processamento de textos, dados científicos e criação de CGIs para páginas dinâmicas para a web. Foi considerada pelo público a 3ª linguagem "mais amada", de acordo com uma pesquisa conduzida pelo site Stack Overflow em 2018,[6] e está entre as 5 linguagens mais populares, de acordo com uma pesquisa conduzida pela RedMonk.[7]

O nome Python teve a sua origem no grupo humorístico britânico Monty Python,[8] criador do programa Monty Python's Flying Circus, embora muitas pessoas façam associação com o réptil do mesmo nome (em português, píton ou pitão).

História

Guido van Rossum, São Francisco, Califórnia

O Python foi concebido no final de 1989[5][8] por Guido van Rossum no Instituto de Pesquisa Nacional para Matemática e Ciência da Computação (CWI), nos Países Baixos, como um sucessor da ABC capaz de tratar exceções e prover interface com o sistema operacional Amoeba[9] através de scripts. Também da CWI, a linguagem ABC era mais produtiva que C, ainda que com o custo do desempenho em tempo de execução. Mas ela não possuía funcionalidades importantes para a interação com o sistema operacional, uma necessidade do grupo. Um dos focos primordiais de Python era aumentar a produtividade do programador.[8]

Python foi feita com base na linguagem ABC, possui parte da sintaxe derivada do C, compreensão de listas, funções anonimas e função map de Haskell. Os iteradores são baseados na Icon, tratamentos de exceção e módulos da Modula-3, expressões regulares de Perl.

Em 1991, Guido publicou o código (nomeado versão 0.9.0) no grupo de discussão alt.sources.[1] Nessa versão já estavam presentes classes com herança, tratamento de exceções, funções e os tipos de dado nativos list, dict, str, e assim por diante. Também estava

presente nessa versão um sistema de módulos emprestado do Modula-3. O modelo de exceções também lembrava muito o do Modula-3, com a adição da opção else clause.[9]Em 1994 foi formado o principal fórum de discussão do Python, comp.lang.python, um marco para o crescimento da base de usuários da linguagem.

A versão 1.0 foi lançada em janeiro de 1994. Novas funcionalidades incluíam ferramentas para programação funcional como lambda, map, filter e reduce. A última versão enquanto Guido estava na CWI foi o Python 1.2. Em 1995, ele continuou o trabalho no CNRI em Reston, Estados Unidos, de onde lançou diversas versões. Na versão 1.4 a linguagem ganhou parâmetros nomeados (a capacidade de passar parâmetro pelo nome e não pela posição na lista de parâmetros) e suporte nativo a números complexos, assim como uma forma de encapsulamento.[10]

Ainda na CNRI, Guido lançou a iniciativa Computer Programming for Everybody (CP4E; literalmente, "Programação de Computadores para Todos"), que visava tornar a programação mais acessível, um projeto financiado pela DARPA.[11] Atualmente o CP4E encontra-se inativo.

Em 2000, o time de desenvolvimento da linguagem se mudou para a BeOpen a fim de formar o time PythonLabs. A CNRI pediu que a versão 1.6 fosse lançada para marcar o fim de desenvolvimento da linguagem naquele local. O único lançamento na BeOpen foi o Python 2.0, e após o lançamento o grupo de desenvolvedores da PythonLabs agrupou-se na Digital Creations.

Python 2.0 implementou list comprehension, uma relevante funcionalidade de linguagens funcionais como SETL e Haskell. A sintaxe da linguagem para essa construção é bastante similar a de Haskell, exceto pela preferência do Haskell por caracteres de pontuação e da preferência do python por palavras reservadas alfabéticas. Essa versão 2.0 também introduziu um sistema coletor de lixo capaz de identificar e tratar ciclos de referências.[12]

Já o 1.6 incluiu uma licença CNRI substancialmente mais longa que a licença CWI que estavam usando nas versões anteriores. Entre outras mudanças, essa licença incluía uma cláusula atestando que a licença era governada pelas leis da Virgínia. A Free Software Foundation alegou que isso era incompatível com a GNU GPL. Tanto BeOpen quanto CNRI e FSF negociaram uma mudança na licença livre do Python que o tornaria compatível com a GPL. Python 1.6.1 é idêntico ao 1.6.0, exceto por pequenas correções de falhas e uma licença nova, compatível com a GPL.[13]

Python 2.1 era parecido com as versões 1.6.1 e 2.0. Sua licença foi renomeada para Python Software Foundation License.[4] Todo código, documentação e especificação desde o lançamento da versão alfa da 2.1 é propriedade da Python Software Foundation (PSF), uma organização sem fins lucrativos fundada em 2001, um modelo tal qual da Apache Software

Foundation.[13] O lançamento incluiu a mudança na especificação para suportar escopo aninhado, assim como outras linguagens com escopo estático.[14] Esta funcionalidade estava desativada por padrão, e somente foi requerida na versão 2.2.

Uma grande inovação da versão 2.2 foi a unificação dos tipos Python (escritos em C) e classes (escritas em Python) em somente uma hierarquia. Isto tornou o modelo de objetos do Python consistentemente orientado a objeto.[15] Também foi adicionado generator, inspirado em Icon.[16]

O incremento da biblioteca padrão e as escolhas sintáticas foram fortemente influenciadas por Java em alguns casos: o pacote logging[17] introduzido na versão 2.3,[18] o analisador sintático SAX, introduzido na versão 2.0 e a sintaxe de decoradores que usa @,[19] adicionadas na versão 2.4.[20]

Em 1 de outubro de 2008 foi lançada a versão 2.6, já visando a transição para a versão 3.0 da linguagem. Entre outras modificações, foram incluídas bibliotecas para multiprocessamento, JSON e E/S, além de uma nova forma de formatação de cadeias de caracteres.[21]

Atualmente a linguagem é usada em diversas áreas, como servidores de aplicação e computação gráfica. Está disponível como linguagem de script em aplicações como OpenOffice (Python UNO Bridge), Blender e pode ser utilizada em procedimentos armazenados no sistema gerenciador de banco de dados PostgreSQL (PL/Python).

A terceira versão da linguagem foi lançada em dezembro de 2008,[22] chamada Python 3.0 ou Python 3000. Com noticiado desde antes de seu lançamento,[23] houve quebra de compatibilidade com a família 2.x para corrigir falhas que foram descobertas neste padrão, e para limpar os excessos das versões anteriores.[8] A primeira versão alfa foi lançada em 31 de agosto de 2007, a segunda em 7 de dezembro do mesmo ano.

Mudanças da versão incluem a alteração da palavra reservada print, que passa a ser uma função, tornando mais fácil a utilização de uma versão alternativa da rotina. Em Python 2.6, isso já está disponível ao adicionar o código from __future__ import print_function.[24] Também, a mudança para Unicode de todas as cadeias de caracteres.[25]

Em 2012, foi criado o Raspberry Pi, cujo nome foi baseado na linguagem Python. Uma das principais linguagens escolhidas é Python. Python influenciou várias linguagens, algumas delas foram Boo e Cobra, que usa a indentação como definição de bloco e Go, que se baseia nos princípios de desenvolvimento rápido de Python.

Atualmente, Python é um dos componentes padrão de vários sistemas operacionais, entre eles estão a maioria das distribuições do Linux, AmigaOS 4, FreeBSD, NetBSD, OpenBSD e OS X. A linguagem se tornou a padrão no curso de ciências da computação do MIT em 2009

Filosofia

Python 3. The standard type hierarchy.png

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)",[26] escrito pelo programador em Python de longa data Tim Peters, descrevendo sumariamente a filosofia do Python. Pode-se vê-lo através de um easter egg do Python pelo comando:

```
>>> import this
```

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.

Tipos de dado

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:

Tipo de dado Descrição Exemplo da sintaxe

str, unicode Uma cadeia de caracteres imutável 'Wikipedia', u'Wikipedia'

list Lista heterogênea mutável [4.0, 'string', True]

tuple Tupla imutável (4.0, 'string', True)

set, frozenset Conjunto não ordenado, não contém elementos duplicados set([4.0, 'string', True])

frozenset([4.0, 'string', True])

dict conjunto associativo {'key1': 1.0, 'key2': False}

int Número de precisão fixa, é transparentemente convertido para long caso não caiba em um int. 42

2147483648L

float Ponto flutuante 3.1415927

complex Número complexo 3+2j

bool Booleano True ou False

!= Diferente Diferente

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.[27]

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 object e são instancias de type). As classes new-styles nada mais são que tipos definidos pelo usuário.

Palavras reservadas

O Python 3 define as seguintes palavras reservadas:[28]

False None True 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

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 tratam "", 0, None, 0.0, [] e {} como falso, enquanto o restante é tratado como verdadeiro de modo geral. Na versão 2.2.1 as constantes

True e False foram adicionadas (subclasses de 1 e 0 respectivamente). 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.

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:

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

Nota: A partir da versão 3.0, o comando print passou a ser uma função, sendo obrigatório o uso de parênteses.[29]

Análise léxica

Exemplo de script

No segundo capítulo do Manual de Referência da Linguagem Python é citado que a análise léxica é uma análise do interpretador em si, 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 ou NOVA LINHA, 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:

```
if resultado > 2 and \
```

```
   1 <= 5  and \
   2 < 5:
      print ('Resultado: %f' % d)
```

ou

```
MESES_DO_ANO = ['janeiro', 'fevereiro', 'março',
            '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 na pilha. 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 (empilhamento). Se o nível de indentação for menor, então é desempilhado até chegar a um nível de indentação recebendo o nome DEDENT (desempilhamento). 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 (Language Reference Manual):

```
       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
```

Indentação

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.

Usando um editor de texto comum é muito fácil existir 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:

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!'
```

O código está correto para os dois exemplos, mas o analisador léxico verificará se a indentação está coerente. O analisador 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: "Seu programa está com erro no bloco de indentação".

Na internet, há uma comparação de velocidade e de codificação entre as linguagens Python e BASIC, esta última, o dialeto BBC BASIC for Windows.

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 novamente o programa, 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. Porém 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. 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. Para descompilar um código de byte é utilizado o módulo dis da biblioteca padrão da linguagem e existem módulos de terceiros que tornam o bytecode mais confuso, tornando a descompilação ineficaz.

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.

Orientação a objetos

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.[30]

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) (ver código contextualizado em anexo).

Em Python não existe proteção dos membros duma classe ou instância pelo interpretador, o chamado encapsulamento. Convenciona-se que atributos com o nome 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 (ver código contextualizado em anexo).

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 (ver código contextualizado em anexo).

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__.

Programação funcional

Uma das construções funcionais de Python é compreensão de listas, uma forma de construir listas. Por exemplo, pode-se usar a técnica para calcular as cinco primeiras potências de dois. O

algoritmo quicksort também pode ser expressado usando a mesma técnica (ver códigos contextualizados para ambos os casos em anexo).

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 Lambda). 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 (ver códigos contextualizados para ambos os casos em anexo). Já geradores foram introduzidos na versão 2.2 e finalizados na versão 2.3, e representam o mecanismo de Python para a avaliação preguiçosa de funções (ver códigos contextualizados para ambos os casos em anexo).

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 apóia 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 (recurso inexistente, permissão de acesso insuficiente, recurso já em uso, ...).

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 termos de desempenho é preferível capturar exceções do que testar atributos antes de os usar. Segue abaixo exemplos de código que testam atributos ("pedem permissão") e que capturam exceções ("pedem perdão"):

Teste de atributo

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

Captura de exceção

```
try:
    ham = spam.eggs
```

```
except AttributeError:
    handle_error()
```

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. Na maioria dos casos o paradigma da captura de exceções é mais rápido, e também pode evitar problemas de concorrência.[31] 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.

Biblioteca padrão

Python possui uma grande biblioteca padrão, geralmente citada como um dos maiores trunfos da linguagem,[32] 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.

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.

Algumas partes da biblioteca são cobertas por especificações (por exemplo, a implementação WSGI da wsgiref segue o PEP 333[33]), mas a maioria dos módulos não segue.

Interoperabilidade

Um 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 que podem ser chamadas diretamente de código Python - mas atualmente esse sequer é o modo mais indicado de interoperação, havendo alternativas tais como Cython, Swig ou cffi. A biblioteca Boost do C++ inclui uma biblioteca para permitir a interoperabilidade entre as duas linguagens, e pacotes científicos fazem uso de bibliotecas de alta performance numérica escritos em Fortran e mantidos há décadas.

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 o métodos de múltiplas linhas em ambos os casos.

Diferente de comentários, a 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().

Plataformas disponíveis

A linguagem e seu interpretador estão disponíveis para as mais diversas plataformas, desde Unix (Linux, FreeBSD, Solaris, MacOS X, etc.), Windows, .NET, versões antigas de MacOS até consoles de jogos eletrônicos ou mesmo alguns celulares, como a série 60, N8xx(PyMaemo) da Nokia e palmtops.

Para algum sistema operacional não suportado, basta que exista um compilador C disponível e gerar o Python a partir do fonte. O código fonte é traduzido pelo interpretador para o formato bytecode, que é multiplataforma e pode ser executado e distribuído sem fonte original.

Implementações

A implementação original e mais conhecida do Python é o CPython, escrita em C e compatível com o padrão C89,[34] sendo distribuída com uma grande biblioteca padrão escrita em um misto de Python e C. Esta implementação é suportada em diversas plataformas, incluindo Microsoft Windows e sistemas Unix-like modernos.

Stackless Python é uma variação do CPython que implementa microthreads (permitindo multitarefa sem o uso de threads), sendo suportada em quase todas as plataformas que a implementação original.

Existem também implementações para plataformas já existentes: Jython para a Plataforma Java e IronPython para .NET.

Em 2005 a Nokia lançou um interpretador Python para os telefones celulares S60, chamado PyS60. Essa versão inclui vários módulos das implementações tradicionais, mas também alguns módulos adicionais para a integração com o sistema operacional Symbian. Uma implementação para Palm pode ser encontrada no Pippy. Já o PyPy, é a linguagem Python totalmente escrita em Python.

Diversas implementações, como CPython, pode funcionar como um interpretador de comandos em que o usuário executa as instruções sequencialmente, recebendo o resultado automaticamente. A execução compilada do código oferece um ganho substancial em velocidade, com o custo da perda da interatividade.

Desenvolvimento

O desenvolvimento de Python é conduzido amplamente através do processo Python Enhancement Proposal ("PEP"), em português Proposta de Melhoria do Python. Os PEPs são documentos de projeto padronizados que fornecem informações gerais relacionadas ao Python, incluindo propostas, descrições, justificativas de projeto (design rationales) e explicações para características da linguagem. PEPs pendentes são revisados e comentados por Van Rossum, o Benevolent Dictator for Life (líder arquiteto da linguagem) do projeto Python. Desenvolvedores do CPython também se comunicam através de uma lista de discussão, python-dev, que é o fórum principal para discussão sobre o desenvolvimento da linguagem. Questões específicas são discutidas no gerenciador de erros Roundup mantido em python.org. O desenvolvimento acontece no auto-hospedado svn.python.org

Licença

Python possui uma licença livre aprovada pela OSI e compatível com a GPL, porém menos restritiva. Ela prevê (entre outras coisas) que binários da linguagem sejam distribuídos sem a necessidade de fornecer o código fonte junto.[4]

Módulos e frameworks

Ao longo do tempo têm sido desenvolvidos pela comunidade de programadores muitas bibliotecas de funções especializadas (módulos) que permitem expandir as capacidades base da linguagem. Entre estes módulos especializados destacam-se:

Descrição Campos de atuação

Django Framework para desenvolvimento ágil de aplicações web; desenvolvimento web

Pylons Framework para desenvolvimento de aplicações web; desenvolvimento web

TurboGears Framework baseado em várias outras tecnologias existentes no mundo que gira em torno da linguagem Python; desenvolvimento web

Matplotlib - Matplotlib / Pylab biblioteca para manipulação de gráficos 2D; processamento de imagem

Python Imaging Library biblioteca para manipulação de imagens digitais; processamento de imagem

PyOpenGL - Python OpenGL Binding suporte multiplataforma ao OpenGL; computação gráfica

Pygame Conjunto de módulos para o desenvolvimento de jogos eletrônicos, incluindo gráficos SDL; desenvolvimento de jogos eletrônicos; computação gráfica

Twisted Framework para o desenvolvimento de aplicações de rede. Inclui módulos para servidor web, de aplicação, SSH e diversos outros protocolos; desenvolvimento de software; desenvolvimento web

PYRO - Python Remote Objects Framework para o desenvolvimento de sistemas distribuídos; computação distribuída

ZODB Sistema de persistência e banco de dados orientado a objetos; banco de dados

Plone SGC - Sistema de gerenciamento de conteúdo; desenvolvimento web

CherryPy Framework para aplicações web; desenvolvimento web

Web2py Framework para aplicações web; desenvolvimento web

Visual Python Framework 3D de alto nível; computação gráfica

SQLObject Mapeador objeto-relacional: traduz estruturas relacionais para objetos Python e manipula o banco de dados de forma transparente; banco de dados

Numarray Módulo para manipulação de vetores e computação científica. computação científica

Interfaces gráficas

Exemplos de bibliotecas de GUI disponíveis para Python incluem:

Descrição

Tkinter Módulo padrão para GUI no Python

PyGTK interface para a biblioteca GTK+

PyQt interface para a biblioteca Qt

wxPython interface para a biblioteca wxWidgets

Etk interface para a biblioteca EFL

Wax Construído para simplificar o uso do wxPython

Kivy Toolkit multiplataforma

Ambientes de desenvolvimento integrado

Existem vários ambientes de desenvolvimento integrado (IDE) disponíveis para Python:

IDE Desenvolvedor Última versão Plataforma Toolkit Licença

IDLE Guido van Rossum et al. Distribuído com CPython Multiplataforma Tkinter PSFL

PyCharm JetBrains 2017.3 (29/11/2017) Java Swing Apache 2.0

Komodo Edit	ActiveState	10 (17/05/2016)	Windows, Linux, macOS	XUL	MPL 1.1
Atom	GitHub	1.25.0 (15/03/2018)	Windows, Linux, macOS	Electron	MIT
GNOME Builder	GNOME Project	3.36.0 (06/03/2020)	Linux	GTK	GNU GPLv3+
Pyzo	Pyzo team	4.4.3 (09/10/2017)	Multiplataforma	PyQt	BSD
Boa Constructor	Team	0.6.1	Independente	wxPython	GNU GPL
Eric Python IDE	Detlev Offenbach	4.1.2	Independente	Qt	GNU GPL
Geany	Team	1.23	Independente	GTK2	GNU GPL
IronPython Studio	Clarius Labs	1.0 (10/122007)	Windows	VS2008 Shell	Runtime Microsoft Public License
PyDev (Eclipse)	Appcelerator	5.7.0 (11/04/2017)	Java	SWT	EPL
PythonCard	Alex Tweedly	0.8.2	Multiplataforma	wxPython	BSD
PyScripter	mmm-experts	1.7.2 (10/2006)	Windows		MIT
Stani's Python Editor	Stani	0.8.4c (14/02/2008)	Independente	wxPython	GNU GPL
Spyder	Spyder developer community	2.3.2 (03/12/2014)	Windows, Linux, macOS	PyQt	MIT
Wing IDE	Wingware	3.0.2-1 (27/11/2007)	Windows, Linux, macOS	PyGTK	Proprietário

Aplicações

Alguns dos maiores projetos que utilizam Python são o servidor de aplicação Zope, o compartilhador de arquivos Mnet, o sítio YouTube e o cliente original do BitTorrent. Grandes organizações que usam a linguagem incluem Google[35] (parte dos crawlers), Yahoo! (para o sítio de grupos de usuários) e NASA.[36] O sistema de gerenciamento de reservas da Air Canada também usa Python em alguns de seus componentes.[37] A linguagem também tem bastante uso na indústria da segurança da informação.

A linguagem tem sido embarcada como linguagem de script em diversos softwares, como em programas de edição tridimensional como Maya,[38] Autodesk Softimage, TrueSpace e Blender.[39] Programas de edição de imagem também a usam para scripts, como o GIMP.[40] Para diversos sistema operacionais a linguagem já é um componente padrão, estando disponível em diversas distribuições Linux. O Red Hat Linux usa Python para instalação, configuração e gerenciamento de pacotes.

Outros exemplos incluem o Plone, sistema de gerenciamento de conteúdo desenvolvido em Python e Zope e a Industrial Light & Magic,[41] que produz filmes da série Star Wars usando extensivamente Python para a computação gráfica nos processos de produção dos filmes.

Referências

«HISTORY». Fonte do Python (em inglês). Python Software Foundation. Consultado em 5 de junho de 2008. Arquivado do original em 17 de fevereiro de 2016

«Python 3.8.3». www.python.org (em inglês). 13 de maio de 2020. Consultado em 25 de junho de 2020

Guido van Rossum (Maio de 1996). «Foreword for "Programming Python" (1st ed.)» (em inglês). Python Software Foundation. Consultado em 12 de junho de 2008

«History and License» (em inglês). Python Software Foundation. Consultado em 7 de abril de 2020

«The Making of Python» (em inglês). Artima Developer. Consultado em 22 de março de 2007

«Stack Overflow Developer Survey 2018». Stack Overflow. Consultado em 16 de abril de 2018

O'Grady, Stephen (7 de março de 2018). «The RedMonk Programming Language Rankings: January 2018» (em inglês). RedMonk. Consultado em 13 de março de 2018

Naomi Hamilton (5 de agosto de 2008). «The A-Z of Programming Languages: Python» (em inglês). Computerworld. Consultado em 17 de agosto de 2008

«Why was Python created in the first place?» (em inglês). Python FAQ. Consultado em 22 de março de 2007

«LJ #37: Python 1.4 Update» (em inglês). Consultado em 29 de abril de 2007. Arquivado do original em 1 de maio de 2007

Guido van Rossum. «Computer Programming for Everybody» (em inglês). Consultado em 22 de março de 2007. Arquivado do original em 23 de fevereiro de 2009

A.M. Kuchling and Moshe Zadka. «What's New in Python 2.0» (em inglês). Consultado em 22 de março de 2007. Arquivado do original em 14 de dezembro de 2009

«History of the software». Referência da Biblioteca Python (em inglês). Consultado em 22 de março de 2007

Jeremy Hylton. «Statically Nested Scopes» (em inglês). Consultado em 22 de março de 2007

«2 PEPs 252 and 253: Type and Class Changes» (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008

«4 PEP 255: Simple Generators» (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008

«PEP 282 - A Logging System» (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008

«8 PEP 282: The logging Package» (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008

«PEP 318 - Decorators for Functions and Methods» (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008

«5 PEP 318: Decorators for Functions and Methods» (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008

A.M. Kuchling (1 de outubro de 2008). «What's New in Python 2.6» (em inglês). Python Software Foundation. Consultado em 3 de outubro de 2008

«Python 3.0 Release» (em inglês). Python Software Foundation. Consultado em 3 de dezembro de 2008

Sarah Stokely (1 de fevereiro de 2008). «Python 3.0 to be backwards incompatible» (em inglês). iTnews. Consultado em 11 de junho de 2008

Georg Brandl. «Make print a function» (em inglês). Consultado em 3 de outubro de 2008

«Diferenças entre Python 2 e Python 3». Blog da Caelum. Consultado em 16 de março de 2019

«PEP 20 - The Zen of Python» (em inglês). Python - Núcleo de Desenvolvimento. Consultado em 15 de janeiro de 2010

«Classes — Random Remarks». Python Documentation (em inglês). Python Software Foundation

DOWNEY, Allen B. Pense em Python. [S.l.]: Novatec. 38 páginas. ISBN 9788575225080

«What's New In Python 3.0» (em inglês). Python Software Foundation. Consultado em 15 de janeiro de 2011

Michele Simionato. «The Python 2.3 Method Resolution Order» (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008

«EAFP vs LBYL (was Re: A little disappointed so far)». web.archive.org. Consultado em 6 de maio de 2012. Arquivado do original em 29 de setembro de 2007

Przemyslaw Piotrowski (Julho de 2006). «Build a Rapid Web Development Environment for Python Server Pages and Oracle» (em inglês). Oracle. Consultado em 11 de junho de 2008

Phillip J. Eby (7 de dezembro de 2003). «PEP 333 -- Python Web Server Gateway Interface v1.0» (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008

Guido van Rossum (5 de julho de 2001). «PEP 7 -- Style Guide for C Code» (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008

«Quotes about Python» (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008

Daniel G. Shafer (17 de janeiro de 2003). «Python Streamlines Space Shuttle Mission Design» (em inglês). Python Software Foundation. Consultado em 11 de junho de 2008

Darryl K. Taft (5 de março de 2005). «Python Slithers into Systems» (em inglês). eWEEK. Consultado em 11 de junho de 2008

«Introduction to Maya Python API». Documentação do Maya (em inglês). Autodesk. Consultado em 18 de julho de 2008

«Python Scripts» (em inglês). Blender. Consultado em 18 de julho de 2008. Arquivado do original em 18 de junho de 2012

James Henstridge (16 de maio de 2006). «GIMP Python Documentation». Documentação do GIMP (em inglês). GIMP. Consultado em 18 de julho de 2008

Robin Rowe (1 de julho de 2002). «Industrial Light & Magic» (em inglês). Linux Journal. Consultado em 18 de julho de 2008

Bibliografia

Pilgrim, Mark (2004). Dive into Python (em inglês) 2 ed. Nova Iorque: Apress. 413 páginas. ISBN 978-1-5905-9356-1

Pilgrim, Mark (2009). Dive into Python 3 (em inglês) 2 ed. Nova Iorque: Apress. 360 páginas. ISBN 978-1-4302-2415-0

Downey, Allen B. (2012). Think Python (em inglês). Sebastopol (Califórnia): O'Reilly. 300 páginas. ISBN 978-1-4493-3072-9

Lutz, Mark (2013). Learning Python (em inglês) 5 ed. Sebastopol (Califórnia): O'Reilly. 1600 páginas. ISBN 978-1-4493-5573-9

Lutz, Mark (2010). Programming Python (em inglês) 4 ed. Sebastopol (Califórnia): O'Reilly. 1632 páginas. ISBN 978-0-596-15810-1

David Beazley e Brian K. Jones (2013). Python Cookbook (em inglês) 3 ed. Sebastopol (Califórnia): O'Reilly. 706 páginas. ISBN 978-1-4493-4037-7

C++

C++ (em português: lê-se "cê mais mais", em inglês lê-se see plus plus) é uma linguagem de programação compilada multi-paradigma (seu suporte inclui linguagem imperativa, orientada a objetos e genérica) e de uso geral. Desde os anos 1990 é uma das linguagens comerciais mais populares, sendo bastante usada também na academia por seu grande desempenho e base de utilizadores.

Bjarne Stroustrup desenvolveu o C++ (originalmente com o nome C with Classes,[4] que significa C com classes em português) em 1983 no Bell Labs como um adicional à linguagem C. Novas características foram adicionadas com o tempo[5], como funções virtuais, sobrecarga de operadores, herança múltipla, gabaritos e tratamento de exceções. Após a padronização ISO realizada em 1998 e a posterior revisão realizada em 2003, uma nova versão da especificação da linguagem foi lançada em dezembro de 2014, conhecida informalmente como C++17[6][7][8][5]

História

A evolução da linguagem

O C++ foi inicialmente desenvolvido por Bjarne Stroustrup dos Bell Labs, durante a década de 1980 com o objetivo implementar uma versão distribuída do núcleo Unix.[4] Como o Unix era escrito em C, deveria-se manter a compatibilidade, ainda que adicionando novos recursos. Alguns dos desafios incluíam simular a infraestrutura da comunicação entre processos num sistema distribuído ou de memória compartilhada e escrever drivers para tal sistema.

Stroustrup percebeu que a linguagem Simula 67 possuía características bastante úteis para o desenvolvimento de software, mas que era muito lenta para uso prático. Por outro lado, a linguagem BCPL era rápida, mas possuía demasiado baixo nível, dificultando sua utilização no desenvolvimento de aplicações. A partir de sua experiência de doutorado, começou a acrescentar elementos do Simula 67 no C, especialmente os recursos de criação e manipulação de objetos. O C foi escolhido como base de desenvolvimento da nova linguagem pois possuía uma proposta de uso genérico, era rápido e também portável para diversas plataformas. Algumas outras linguagens que também serviram de inspiração para o cientista da computação foram ALGOL 68, Ada, CLU e ML.

Bjarne Stroustrup em 2007

Ainda em 1983 o nome da linguagem foi alterado de C with Classes para C++. Antes implementada usando um pré-processador, a linguagem passou a exigir um compilador próprio, escrito pelo próprio Stroustrup.[4] Novas características foram adicionadas, como funções virtuais,[4] sobrecarga de operadores e funções,[4] referências, constantes, gerenciamento manual de memória, melhorias na verificação de tipo de dado e estilo de comentário de código de uma linha (//). Em 1985 foi lançada a primeira edição do livro The C++ Programming Language, contendo referências para a utilização da linguagem, já que ainda não era uma norma oficial. A primeira versão comercial foi lançada em outubro do mesmo ano.[9] Em 1989 a segunda versão foi lançada, contendo novas características como herança múltipla, classes abstratas, métodos estáticos, métodos constantes e membros protegidos, incrementando o suporte a orientação a objeto. Em 1990 foi lançado o livro The Annotated C++ Reference Manual, que tornou-se base para o futuro padrão. Outras adições na linguagem incluem gabaritos, tratamento de exceções, espaço de nomes, conversão segura de tipo de dado e o tipo booleano.

Assim como a linguagem, sua biblioteca padrão também sofreu melhorias ao longo do tempo. Sua primeira adição foi a biblioteca de E/S, e posteriormente a Standard Template Library (STL); ambas tornaram-se algumas das principais funcionalidades que distanciaram a linguagem em relação a C. Criada primordialmente na HP por Alexander Stepanov[10] no início da década de 1990 para explorar os potenciais da programação genérica, a STL foi apresentada a um comitê unificado ANSI e ISO em 1993 à convite de Andrew Koenig. Após uma proposta formal na reunião do ano seguinte, a biblioteca recebe o aval do comitê.

Depois de anos de trabalho, o mesmo comitê ANSI/ISO padronizou o C++ em 1998 (ISO/IEC 14882:1998). Após alguns anos foram reportados defeitos e imprecisões no documento, e uma correção foi lançada em 2003.[11]

Por muito tempo, o C++ foi encarado como um superconjunto do C.[nota 1] Entretanto, em 1999 o novo padrão ISO para a linguagem C tornou as duas linguagens ainda mais diferentes

entre si. Devido a essas incompatibilidades, muitas empresas que desenvolvem compiladores não oferecem suporte à versão mais recente da linguagem C.

Pode-se dizer que C++ foi a única linguagem entre tantas outras que obteve sucesso como uma sucessora à linguagem C, inclusive servindo de inspiração para outras linguagens como Java, a IDL de CORBA e C#.

Etimologia

Durante sua fase inicial de desenvolvimento, a linguagem era chamada "novo C", "C84" ou ainda "C com classes".[4] O termo "C++" é creditado a Rick Mascitti,[12] e foi utilizado pela primeira vez em dezembro de 1983. O termo é uma referência ao operador de incremento ++, significando um acréscimo (uma evolução) à linguagem C. Em tom humorado, desenvolvedores de software e especialistas em informática no início da década de 1990 costumavam relacionar o ++ do nome à grande insistência dos programadores em utilizar o C++ da mesma forma que a linguagem C, não usufruindo das novas facilidades que a linguagem poderia fornecer. Assim como o ++ estava sendo aplicado de maneira pós-fixa à letra C, a linguagem C++ era uma evolução do C pós-fixada, que só tornar-se-ia realidade em algum futuro remoto, não naquele momento.

Trabalhos futuros

A linguagem continua evoluindo de forma a fornecer novas funcionalidades. O grupo de desenvolvimento Boost.org trabalha para evoluir a biblioteca padrão, informando o comitê oficial da linguagem das facilidades que possuem maior retorno positivo dos programadores, seja por qualidade ou por utilidade, e quais ainda devem ser desenvolvidas. Tudo indica que o C++ continuará com sua natureza multiparadigma. Por exemplo, o trabalho da Boost.org dedica-se a acrescentar as qualidades da programação funcional e genérica. O padrão C++ não define a implementação para a definição de nomes e tratamento de exceções, entre outras facilidades específicas, o que frequentemente torna incompatíveis códigos objeto produzidos por diferentes compiladores. Apesar disso, existem padrões periféricos específicos para certas plataformas ou sistemas operacionais para padronizar compiladores dessas plataformas, como por exemplo o C++ ABI.[nota 2]

As empresas de desenvolvimento de compiladores ainda se esforçam para suportar inteiramente o padrão, especialmente na área de gabaritos. Uma das disputas se refere à palavra reservada export, que permite que a definição de um gabarito seja separada de sua declaração. O primeiro compilador a implementar export foi o Comeau C++ em 2003 (cinco anos após o lançamento do padrão), e no ano seguinte uma versão beta do Borland C++ Builder X também suportava a facilidade. Interessante notar que ambos os compiladores são baseados na versão EDG do C++. Muitos livros fornecem exemplos de códigos para implementar export[nota 3] que não são compiláveis, mas não há referências para o problema mencionado. Outros compiladores como o Microsoft Visual C++ e o GCC não suportam a facilidade. O secretário do comitê oficial do C++ Herb Sutter recomendou que a palavra fosse

removida de versões futuras do padrão da linguagem,[13] mas após discussão a decisão final foi mantê-la.[14]

Outras disputas relativas a gabaritos se referem à especialização parcial, que foi pouco suportada por muitos anos depois que o C++ padrão foi lançado.

Atualmente a linguagem tem uma nova especificação, conhecida por C++11 e publicada como 14882:2011.[15]

Características

História descritiva

No livro In The Design and Evolution of C++ (1994), Bjarne Stroustrup descreve algumas regras que ele utiliza para desenvolver a linguagem, como exemplificado abaixo:

C++ é desenvolvido para ser uma linguagem tipada estaticamente e de propósito geral que é tão eficiente e portátil quanto o C.

C++ é desenvolvido para suportar múltiplos paradigmas.

C++ é desenvolvido para fornecer ao programador escolhas, mesmo que seja possível ao programador escolher a opção errada.

C++ é desenvolvido para ser o mais compatível com C possível, fornecendo transições simples para código C.

C++ evita fornecer facilidades que são específicas a certas plataformas ou a certos grupos de desenvolvedores.

C++ não exige overhead para facilidades que não são utilizadas.

C++ é desenvolvido para ser utilizado mesmo sem um ambiente de desenvolvimento sofisticado.

Stanley B. Lippman documenta em seu livro Inside the C++ Object Model (1996)[16] como compiladores convertem código de programas C++ em mapeamentos de memória. Lippman trabalhou implementando e mantendo o C-front, a implementação original do C++ nos Bell Labs.

Stroustrup sempre desejou que o C++ fosse mantido como uma linguagem de especificação pequena, apesar de pressões externas para adições de novas funcionalidades na especificação da própria linguagem ao invés da codificação de novas bibliotecas para a biblioteca padrão. Brian Kernighan notou que enquanto em C existe geralmente uma maneira de resolver problemas, em C++ existem várias. Na maioria das linguagens de programação, um padrão ou um conjunto bastante restrito de padrões de projeto de software é escolhido para o

desenvolvimento. Entretanto, isso não acontece em C++, pois a escolha é delegada ao desenvolvedor. É um conceito que prega que não existe paradigma de programação ou padrão de desenvolvimento que resolva todos os problemas, por isso a pluralidade e generalidade de aplicações para a linguagem. Tal filosofia assusta iniciantes e professores, que sentem que a linguagem deveria ser de fácil aprendizado, algo que o C++ não é.[carece de fontes]

Biblioteca padrão

Ver artigo principal: Biblioteca padrão do C++

Ver artigo principal: Standard Template Library

A biblioteca padrão do C++ incorpora a biblioteca padrão do C com algumas pequenas modificações para trabalhar melhor com as novas funcionalidades criadas pela linguagem. Outra grande parte da biblioteca é composta pela biblioteca padrão de gabaritos (STL). Ela fornece ferramentas úteis como containers (vetores, listas, entre outros), algoritmos (filtragem de elementos de container, busca, ordenação, entre outros) e iteradores (ponteiros inteligentes genéricos para acessar tais containers e interligá-los aos algoritmos). Usando gabaritos é possível escrever algoritmos genéricos que funcionam para qualquer container ou sequência definida por iteradores. Tendo em vista que um iterador nada mais é que um ponteiro encapsulado, é possível também utilizar os algoritmos genéricos em vetores C, utilizando-se ponteiros comuns para tal. Como em C, os arquivos cabeçalho são incluídos utilizando a diretiva #include. Ao todo são fornecidos 69 arquivos cabeçalho padrão, dos quais 19 deles estão em depreciação.[carece de fontes]

Devido ao fato da biblioteca padrão ter sido desenvolvida por especialistas e de já ter sido amplamente utilizada comercialmente e academicamente, é recomendado utilizar seus componentes ao invés de componentes próprios. Por exemplo, utilizar std::vector e std::string ao invés de declarar vetores herdados do C não somente torna o desenvolvimento mais simples, como também traz mais segurança e escalabilidade para o sistema.

A biblioteca STL foi originalmente desenvolvida pela HP[10] e posteriormente pela SGI, antes de sua incorporação na biblioteca padrão do C++. O padrão não a define como "STL", mas ainda utiliza-se esse termo para distingui-la do resto da biblioteca. O projeto STLPort mantém uma implementação atualizada da biblioteca, e é baseado na SGI STL. O projeto Boost fornece elementos adicionais à STL, dos quais alguns já são considerados a serem parte da biblioteca padrão no futuro.

Operadores

Ver artigo principal: Operadores em C e C++

Os operadores em C++ são um conjunto de todos os operadores do C mais novas adições à linguagem. Um grupo de novos operadores do C++ são os relativos à conversão de tipo de dado, e consistem em const_cast, static_cast, dynamic_cast e reinterpret_cast. Eles são uma evolução a conversão de dados utilizada em C, que limitava-se a oferecer um método para

conversão tal qual static_cast. dynamic_cast refere-se diretamente ao suporte de herança e polimorfismo oferecido pela linguagem, e está relacionado a outro novo operador, typeid, que retorna informações sobre o tipo de dado derivado pelo operando. Ambos os operadores requerem a habilitação de RTTI para funcionar. Outro grupo de novos operadores são os relativos à alocação de memória, e consistem em new e delete. Assemelham-se às funções malloc e free respectivamente, que estão presentes na biblioteca padrão do C. Outro novo operador é o de resolução de âmbito, ::, e que refere-se diretamente ao suporte de espaço de nomes e orientação a objeto oferecido pela linguagem. Com ele é possível declarar e acessar espaços de nomes, e também declarar classes e acessar objetos.

O C++ define que alguns dos operadores podem ser sobrecarregados, o que permite, assim como na sobrecarga de funções, que diferentes tipos de dados sejam passados para um operador de forma a produzir diferentes resultados. Essa técnica também permite que classes definidas por utilizadores também possam usufruir de operadores próprios, tornando possível que uma classe Lista possa sobrecarregar o operador de apêndice += para que diversos elementos possam ser adicionados a lista, como elementos ou outras listas. Alguns operadores de classes definidas pelo utilizador devem ser obrigatoriamente sobrecarregados (definidos) a fim de poderem ser utilizados pela STL. Por exemplo, uma classe Funcionario deve fornecer o operador menor que (<) para ser utilizada pela função de ordenação (sort). De acordo com o padrão atual da linguagem, este requerimento é implícito durante a compilação: caso a função sort seja invocada para a um container da classe Funcionario e esta não define o operador <, há erro de compilação. Para padrões futuros planeja-se introduzir os "conceitos", que auxiliaram a programação genérica na especificação dos requerimentos de um tipo de dado para que ele seja usado em uma função. Por exemplo, os iteradores passados para sort estarão associados ao conceito "tipo de dado comparável", isto é, um tipo de dado que declara o operador <. Ao explicitar essa relação o código se torna mais consistente, e o compilador é auxiliado a fim de retornar uma mensagem de erro mais adequada ao utilizador caso haja problemas durante a compilação.

Pré-processador

Ver artigo principal: Pré-processador

O C++ é compilado em três fases: pré-processamento, compilação propriamente dita (tradução para código objeto) e ligação.[17][18] Durante a primeira fase, as diretivas de pré-processamento são aplicadas através de transformações léxicas no próprio código fonte, que então alimenta as próximas fases de compilação. Elas são identificadas no código através do caractere #. O pré-processamento é utilizado para substituir partes de código, para inutilizar partes de código e para importar módulos externos.

Por exemplo, o código #define PI 3.14159265358979323846264433 fará com que sempre que PI aparecer no código, este será substituído por 3.14159265358979323846264433 e isso ocorre antes de começar a compilar, é como se o usuário tivesse digitado o valor de PI ao invés do texto PI. Outro uso do pré-processador é o que segue: #include <iostream> fará com que seja incluído (importado) todo o conteúdo da cabeçalho iostream da biblioteca libc.

Gabaritos[vago]

Um gabarito[19], mais conhecido como “template”, é um mecanismo que permite a criação de classes ou funções genéricas, passando tipos de dados como parâmetros. O prefixo template é uma palavra reservada que estabelece que um gabarito está sendo criado, junto a ele vem a expressão <class T> onde T representa um tipo de dado dentro desse gabarito, T pode ser utilizado como um tipo de dado apenas dentro do escopo definido pelo prefixo template <class T>.

Exemplo de um gabarito:

```
#include <iostream>

template <class T>
T min(T a, T b){
        return a<b ? a : b;
}
```

Um gabarito é semelhante a um macro, embora tenham algumas diferenças como, por exemplo, o fato de um gabarito não se restringir a uma substituição léxica, um outro exemplo é que os macros não verificam os tipos de dados dos parâmetros, fazendo com que erros como a falta de compatibilidade entre duas variáveis aconteçam. Quando um gabarito é chamado pela primeira vez para um tipo T, o compilador cria uma nova instancia do gabarito, que será utilizada sempre que o mesmo for chamado com um argumento do tipo T. Para instanciar uma classe genérica[20] é necessário passar como argumento um tipo de dado para o gabarito, como mostrado no exemplo abaixo, onde a classe Pilha recebe como argumento int. Gabaritos são muito uteis para a criação de coleções de classes. Eles permitem a criação de classes genéricas passando um tipo de dado como parâmetro para que seja construída uma cópia da classe para um tipo especifico de dado, como por exemplo, a criação de uma pilha de objetos ou uma pilha de caracteres, fazendo com que não seja necessário criar varias classes Pilha manualmente.

```
#include <iostream>

 int main()
{
   Pilha<int> p(); //
}
```

Objetos

Ver artigo principal: Orientação a objetos

O C++ introduziu alguns conceitos de orientação a objetos ao C, como exemplificado pelas classes, que apresentam quatro características comumente presentes em linguagens de programação orientadas a objeto: abstração, encapsulamento, herança e polimorfismo. Cada vez que uma classe é instanciada é criado um objeto na memória, que é basicamente um conjunto de atributos e operações reunidos.

Encapsulamento

Ver artigo principal: Encapsulamento

O encapsulamento permite que os atributos de classes possam ser declarados como públicos, privados ou protegidos. Um atributo público (o menos restrito) pode ser acessado a partir de qualquer método que tenha acesso ao objeto. Um atributo privado (o mais restrito) só pode ser acessado por métodos da própria classe e por métodos explicitamente declarados como permitidos para tal (utilizando a palavra reservada friend). Atributos protegidos só podem ser acessados por métodos da mesma classe, por métodos de classes herdadas e por métodos explicitamente declarados (utilizando a palavra reservada friend). É considerado como uma boa prática de programação restringir ao máximo o acesso aos atributos, de forma a isolar detalhes de implementação de uma classe, tornando públicas somente as funções membro que realizam uma interface mínima da classe com outros componentes.

O isolamento dos dados proposto pelo encapsulamento não é infalível, podendo ser contornado ao realizar operações de baixo nível em objetos. Dessa maneira, um atributo privado pode ser acessado e modificado a partir de um ponteiro para seu endereço de memória sem problemas, ainda que isso seja considerado uma má prática de programação. Tal característica, herdada da linguagem C, é reflexo direto da liberdade que o C++ fornece ao desenvolvedor em relação a padrões de projeto de software, cabendo a ele decidir qual é o mais adequado para seu algoritmo. O desenvolvedor tanto pode esquecer tal característica, atendo-se somente a detalhes de especificação em alto nível, quanto adequar tais características em sua especificação de forma mais baixo nível, visando desempenho ou algum outro objetivo.

Herança

Ver artigo principal: Herança

A herança de uma classe para com outra pode ser declarada como pública, protegida ou privada. Isso determina o quão relacionadas as classes serão entre si. Somente a herança pública corresponde ao conceito usual de herança, pois permite acesso total aos atributos da classe-base. Entretanto, pode-se também declarar heranças protegidas e privadas, com características parecidas como as detalhadas anteriormente sobre encapsulamento. Essa funcionalidade adicionou ao C++ a possibilidade de criação de classes abstratas, que não podem ser instanciadas, mas que oferecem interfaces de funcionamento para suas respectivas

classes herdadas. Ela é um princípio básico da orientação a objeto para a reutilização de código, pois permite que classes (possivelmente escritas por outros desenvolvedores e então não modificáveis) possam ser herdadas de forma a ser incrementadas em funcionalidade.

A herança múltipla é uma das características do C++ mais controversas. Ela permite que uma classe possa ser derivada de mais de uma classe base, o que pode resultar em um complicado grafo de herança e relacionamento entre classes. Por exemplo, uma classe Gato voador pode ser derivada tanto das classes Gato quanto Mamífero voador. A mistura de heranças reflete em uma mistura de espaços de nomes na classe herdada, o que pode ser resolvido através da declaração local de espaço de nomes, como explicado adiante.

Polimorfismo

Ver artigo principal: Polimorfismo

Polimorfismo é a capacidade de usar um operador ou uma função de diferentes maneiras, permitir fornecer diferentes significados de acordo com o contexto. Uma vez que um aplicativo é escrito utilizando o conceito de polimorfismo, pode ser facilmente estendido, oferecendo novos objetos que estejam em conformidade com a interface original. Não é necessário recompilar programas originais, adicionando novos tipos. Apenas a re-vinculação é necessária para expor as novas mudanças juntamente com a antiga aplicação. Ele auxilia na reutilização de código e contribui para a criação de aplicações robustas. C++ suporta diversos tipos de polimorfismos, sejam estáticos (resolvidos em tempo de compilação de código) ou dinâmicos (resolvidos em tempo de execução de código).

Estático

A sobrecarga de funções é um polimorfismo estático que permite que um programa possa declarar várias funções com o mesmo nome, diferenciando entre si pela quantidade de parâmetros apresentados e por seus respectivos tipos de dado. Assim, o mesmo nome de função pode referir-se a diferentes funções dependendo do contexto em que ela é usada. O tipo retornado pela função não é utilizado para distinguir funções sobrecarregadas. A sobrecarga de operadores também é um polimorfismo estático que permite que a definição de certos operadores resultem em uma chamada de função que depende dos tipos de dado dos operadores sendo utilizados.

O C++ também suporta argumentos padrão para os parâmetros das funções, o que permite omitir tal parâmetro na invocação da função. Quando uma função é chamada com menos argumentos que o esperado e os argumentos explícitos são compatíveis com os parâmetros da esquerda à direita, os últimos parâmetros são atribuídos de acordo com o argumento padrão. Semanticamente parecida com a sobrecarga de funções, essa técnica permite simplificar situações em que uma função é declarada somente para invocar uma sobrecarga dela própria com algum parâmetro especificado.

O C++ implementa polimorfismo paramétrico através de gabaritos, que fazem o compilador gere uma instância separada da classe ou função usada como gabarito para cada permutação de parâmetros de tipo usado com ele, o que pode levar a dificultar a depuração de código. Um benefício que os gabaritos C++ têm sobre Java e C# é permitir a metaprogramação por gabaritos, uma forma de pré-avaliação de parte do código em tempo de compilação ao invés de tempo de execução.

Os gabaritos em C++ fornecem um mecanismo sofisticado para a criação de código genérico polimórfico. Em particular, por meio da técnica Curiously Recurring Template Pattern é possível implementar uma forma de polimorfismo estático que imita a sintaxe para substituir as funções virtuais. Uma vez que gabaritos são sensíveis aos tipos de dados e também são Turing completos, também podem ser usados para permitir que o compilador resolva condicionais recursivas e gerar programas substanciais através de metaprogramação por gabaritos.

Entre os usos de polimorfismo estático, inclui-se funções com o mesmo nome mas que tratam de diferentes parâmetros, como soma(int, int) e soma(double, double) (o que, entretanto, ignora as facilidades dos gabaritos.) Também, versões novas da mesma função que recebem parâmetros adicionais, como ExportarDados(void* buffer, int tamanho) e ExportarDados(void* buffer, int tamanho, unsigned long opcoes). Mais um uso é um mesmo nome de método para atribuir ou obter o valor de uma propriedade, como Classe::Propriedade(int x) e int x Class::Property() const.

Dinâmico

O polimorfismo por herança é um exemplo de polimorfismo dinâmico no qual ponteiros de uma classe base podem referenciar objetos de classes derivadas, o que permite que uma chamada de função virtual seja resolvida em tempo de execução de código. Ponteiros e referências de uma classe base podem referenciar objetos de qualquer classe derivada de si, o que permite que arranjos e outros containers de um dado tipo possam armazenar ponteiros de diversos tipos de dados, o que não poderia ser feito de outra maneira em C++. Como não é possível descobrir se a conversão do tipo base para o tipo derivado é segura em tempo de compilação, a verificação deve ser feita durante a execução do código. Para isso é fornecido o operador dynamic_cast, que permite tentar a conversão segura de uma classe mais abstrata (classe base) para outra mais específica (classe derivada). Para sua utilização a linguagem dispõe do RTTI, uma técnica para manter em memória informações sobre o tipo de dado de objetos. Caso a conversão não seja possível uma exceção específica é lançada.

Normalmente, quando uma função em uma classe derivada substitui uma função em uma classe base, a função chamada é determinada pelo tipo do objeto. Uma dada função é sobrescrita quando não existe nenhuma diferença no número ou tipo de parâmetros, entre duas ou mais definições para aquela função. Assim, em tempo de compilação pode não ser possível determinar o tipo do objeto e, portanto, a função a ser chamada, tendo apenas um ponteiro de classe base, a decisão é adiada até o tempo de execução. Isso é chamado de

despache dinâmico. Funções ou métodos virtuais permitem a implementação mais específica da função a ser chamada, de acordo com o tipo do objeto em tempo real. Em C++, isto é geralmente feito usando tabelas de funções virtuais. Se o tipo de objeto é conhecido, isso pode ser contornado, antecipando o nome da classe antes da chamada de função, mas, em geral chamadas de funções virtuais são resolvidas em tempo de execução.

Além das funções de membro padrão, sobrecargas do operador e destrutores podem ser virtuais. A regra geral é que, se todas as funções da classe são virtuais, o destrutor também deve ser assim. Como o tipo de criação de um objeto é conhecido em tempo de compilação, construtores de cópia e de extensão, não pode ser virtuais. No entanto pode acontecer de uma cópia de um objeto ser criado quando um ponteiro para um objeto derivado é passado como um ponteiro para um objeto base. Nesse caso, uma solução comum é criar um clone() (ou similar) e declarar que a função como virtual. O método clone() cria e retorna uma cópia da classe quando chamado.

Um membro da função também pode ser declarado puramente virtual, acrescentando = 0 após o parêntese de fechamento e antes do ponto e vírgula. Os objetos não podem ser criados de uma classe com uma função virtual pura e são chamados de tipos de dados abstratos. Esses tipos de dados abstratos só podem ser derivados. Qualquer classe derivada herda a função virtual pura deve apresentar uma definição não pura (e todas as outras funções virtuais puras), antes de objetos da classe derivada poderem ser criados.

Regra dos três

Uma regra informal no desenvolvimento orientado a objeto em C++ afirma que se uma classe ou estrutura possui um dos seguintes itens, ela provavelmente deveria ter todos os três:[21] destrutor, construtor de cópia e operador de atribuição (=).

Esse três métodos são funções membros especiais criadas pelo compilador automaticamente se não são definidas pelo desenvolvedor.[nota 4] Se um desses métodos é definido explicitamente pelo desenvolvedor, isso significa que a versão gerada pelo compilador não serve para um dos casos, e portanto muito provavelmente também não serve para os outros casos.

Um adendo a essa regra diz respeito à técnica RAII:[22] se ela for usada então o destrutor pode ser deixado sem definição (também conhecida como "regra dos dois"[23]).

Tratamento de exceções

Ver artigo principal: Tratamento de exceções

O tratamento de exceção é um mecanismo desenvolvido para lidar com a ocorrência de algumas condições (chamadas exceções) que alteram o funcionamento normal do fluxo de um

programa de computador. O C++ suporta tal tratamento, de forma que o estado atual de um programa após uma exceção é alterado automaticamente para outro estado pré-definido para a recuperação do sistema.

Para isso foram adicionadas à linguagem as palavras reservadas try e catch. A primeira especifica um bloco de código que será vigiado em relação à exceções, de forma que se uma for identificada, o fluxo de programa será desviado para um bloco especificado pela segunda palavra reservada. Para um dado bloco try podem existir diversos blocos catch, capturando exceções de diferentes tipos de dado. Alternativamente, a sintaxe catch(…) foi introduzida para especificar um bloco de tratamento de exceção independente do tipo da exceção, genérico.

O conceito puro da ciência da computação para tratamento de exceções ainda inclui o bloco de instruções finally, que indica um bloco de código executado após um bloco try caso nenhuma exceção tenha sido lançada, indicando sucesso na operação. Tal abordagem não foi adicionada ao C++, sendo substituível por outras técnicas como RAII[22][24].

Espaço de nomes

Ver artigo principal: Espaço de nomes

O C++ introduziu os espaços de nomes para a organização das bibliotecas, e sua função é agrupar um contexto para identificadores (variáveis, funções, classes, estruturas, entre outros). No contexto de sistemas operativos, o espaço de nomes poderia ser representado por diretórios. Toda a biblioteca padrão está contida no espaço de nomes std (abreviação de standard, que em inglês significa padrão). Para utilizar um espaço de nomes pode ser feita tanto uma declaração global dos espaços quanto local. Uma declaração global é normalmente inserida no início dos módulos, após a importação dos módulos externos, utilizando a palavra reservada using (como em using namespace std;, ver exemplo contextualizado em anexo). Ela também pode ser usada em um âmbito pré-determinado por um bloco de código. Uma declaração local é inserida antes de invocar o identificador envolvido, utilizando o operador de resolução de âmbito :: (como em std::cout, ver exemplo contextualizado em anexo). A declaração global é útil para reduzir a quantidade de código produzido, sub entendendo a origem dos identificadores utilizados em todo um âmbito. Apesar disso, ela deixa margem à ambiguidades, pois é possível que um mesmo identificador esteja presente em mais de um espaço de nome importado no módulo. Para eliminar esse problema deve-se, além de utilizar a declaração global, declarar o identificador ambíguo localmente cada vez que ele for utilizado.

Em determinadas ocasiões, espaços de nome não considerados durante a primeira verificação do espaço de nomes de uma função podem também ser utilizados na busca, dependendo dos tipos de dados utilizados nos argumentos. A técnica, chamada busca de nomes dependente de argumento ou Koening lookup, ocorre quando a busca explícita pela função não encontra correspondente, começando então a procurar por espaços de nomes associados. Um padrão

muito utilizado pela Standard Template Library é declarar sobrecarga de operadores que somente são encontrados pelo compilador através dessa técnica.

Ponteiros e referências

Ver artigos principais: Ponteiro (programação) e Referência (ciência da computação)

O C++ herdou a funcionalidade de ponteiros do C e toda a aritmética de ponteiros disponível para aquela linguagem: tratando um ponteiro como um tipo inteiro é possível mover-se facilmente por regiões de memória. A instância de um ponteiro em C++ é uma variável que armazena um endereço de memória, e que pode ser nula. A biblioteca padrão ainda fornece auto_ptr, uma espécie de ponteiro inteligente para contagem de referências que pode ser utilizado em algumas situações como uma alternativa segura aos ponteiros primitivos do C, automatizando o processo de desalocação de memória do objeto apontado pelo ponteiro.

Por questões de segurança, o C++ introduziu também o tipo de dado referência, um tipo mais restrito de ponteiro. Uma referência definida para outro objeto não pode ser mais referenciada, qualquer ocorrência do nome no código diz respeito ao objeto referenciado. Como consequência, não é possível realizar "aritmética de referências". Outra consequência é que não é possível alterar uma referência para que ela defina outro objeto; após definida, essa relação vale para todo o tempo de vida. Em contrapartida, um mesmo ponteiro frequentemente aponta para diferentes áreas de memória.

Apesar de teoricamente possível[nota 5], a existência de referências nulas não é considerada, podendo-se assumir que uma referência sempre indica um objeto válido em memória.

Incompatibilidade com C

É incorreto considerar o C++ como um super conjunto de C, isto é, uma linguagem que implementa o C completamente e que adiciona novas funcionalidades.[25] Grande parte de código C pode ser perfeitamente compilado em C++, mas existem algumas pequenas diferenças sintáticas e semânticas entre as linguagens que tornam alguns trechos de código C válidos em código C++ inválido, ou códigos que exibem comportamentos diferentes em cada linguagem.[26]

Talvez a diferença mais comum é que C permite a conversão implícita entre o tipo de dado void* para ponteiros para outros tipos, algo que o C++ não permite. Logo, o seguinte código em C é válido:

```
int *i = malloc(sizeof(int) * 5);        /* conversão implícita de void* para int* */
```

Para assegurar-se que o código funcione tanto em C quanto C++ é necessário explicitar a conversão, acrescentando o que é chamado de "cast":

```
int *i = (int *) malloc(sizeof(int) * 5); /* conversão explícita de void* para int* */
```

Outra questão de portabilidade entre as linguagens é o fato do C++ adicionar várias novas palavras reservadas, como new e class, que podem ser utilizadas como identificadores (por exemplo nomes de variáveis) em C, gerando incompatibilidade.

Algumas outras incompatibilidades foram removidas no padrão C99, que agora suporta facilidades como comentários por //.[26] Tanto C99 quanto C++ definem o tipo de dado bool e suas respectivas constantes true e false. Apesar disso, enquanto a definição no C++ é embarcada na própria linguagem, tornando tais elementos palavras reservadas, em C tais identificadores são declarados através da biblioteca padrão stdbool.h.

Algumas construções sintáticas são válidas tanto em C quanto C++, mas produzem resultado diferente. Por exemplo, o valor literal 'a' possui tipo de dado int em C e char em C++, o que significa que uma chamada sizeof('a'), que retorna a quantidade de bytes ocupada pelo identificador, pode resultar em resultados diferentes entre as duas linguagens. Na prática, esse exemplo específico não é realmente um problema já que caracteres literais são convertidos para o tipo int implicitamente pelo compilador tanto em C quanto em C++.

Análise sintática do código fonte

Como a gramática C++ é bastante complexa, é difícil construir um bom analisador sintático para código fonte C++ utilizando algoritmos clássicos como o LALR(1).[27] Por exemplo, com o LALR é possível analisar código Java.[28] A flexibilidade da linguagem também é fruto de ambiguidades que um analisador simples não consegue distinguir. Por esse motivo, existem poucas ferramentas para análise e transformação não trivial de código, como refatoração. Em suas primeiras especificações o C++ era uma gramática LALR, entretanto, com a adição de funcionalidades como espaço de nomes, exceções, gabaritos e o tipo bool, essa característica logo se tornou inválida.[27]

A análise sintática não é a tarefa mais difícil na construção de uma ferramenta C++. Outras tarefas incluem entender o significado dos identificadores do programa que um compilador deve possuir. Sistemas práticos de processamento não devem somente analisar o código fonte, mas também compreender exatamente cada identificador em diferentes usos, lidando com as regras de âmbito de identificadores (ver um código contextualizado em anexo).

O grupo Boost possui um projeto para a construção de um analisador sintático C++ com base na biblioteca Spirit,[29] de forma que se torna um método padrão e livre de análise sintática para a linguagem.

Exemplos de código

Ver também: Lista de exemplos de código C++

Exemplo de um programa que imprime na tela "Olá, Mundo!".

```
# include <iostream>

int main()
{
   std::cout << "Olá, Mundo!" << std::endl;
   return 0;
}
```

Críticas

Wikiquote

O Wikiquote possui citações de ou sobre: Bjarne Stroustrup

“ C faz com que dar um tiro no pé seja fácil; C++ torna isso mais difícil, mas quando nós o fazemos arrebentamos com a perna toda. ”

A citação de Stroutrup trata com humor o fato de o C++, ao possibilitar a programação de alto nível, ter facilitado a codificação de algoritmos e organização de projetos em relação ao C, uma linguagem que requer constante atenção contra erros lógicos de programação devido à sua alta flexibilidade. Por outro lado, o C++ possui nuances da sintaxe e semântica da linguagem muito sutis, difíceis de serem identificados, e que quando não percebidos podem levar a comportamentos indesejados no código.

Pontos positivos do C++ incluem a produção de código o quanto mais eficiente possível, fazendo uso de otimizações agressivas e sofisticadas dos compiladores. Entretanto, a linguagem ainda mantém alta flexibilidade, portabilidade e consistência. Um exemplo da flexibilidade é a possibilidade de programação de alto e baixo nível. Outro exemplo é a possibilidade de metaprogramação e programação genérica. Mais um ponto positivo é ampla disponibilidade e suporte, devido principalmente à grande base de desenvolvedores. Além disso, a compatibilidade com C resulta em vasta base de código. Fora do lado técnico, um positivo é que a linguagem não está sob o domínio duma empresa (em contraste do Java — Sun Microsystems (atualmente pertencente a Oracle Corporation) ou Visual Basic — Microsoft); a padronização é responsabilidade da ISO.

Por outro lado, pontos negativos do C++ incluem grande período para o aprendizado devido à complexidade da linguagem. Também, os compiladores atuais, que nem sempre produzem o código mais otimizado, tanto em velocidade quanto tamanho do código, e que geralmente produzem mensagens de erro difíceis de serem interpretadas em problemas relacionados com gabaritos. Ainda relacionado a aprendizado e uso, a biblioteca padrão não cobre áreas

importantes da programação, como threads (suportada por C++11), conexões TCP/IP, interface gráfica e manipulação de sistemas de arquivos, o que implica na necessidade de criação de bibliotecas próprias para tal, que pecam em portabilidade e padronização. Apesar da ampla base de código legada do C, o C++ também herdou daquela linguagem problemas de entendimento de sintaxe do mesmo. Mais um ponto negativo é que devido à grande flexibilidade no desenvolvimento, é recomendado o uso de padrões de programação mais amplamente que em outras linguagens.

Comunidade de desenvolvimento

Pessoas notáveis

O desenvolvimento da linguagem C++ é fruto do trabalho de milhares de pessoas associadas à academia e à indústria de software, e pode consistir na utilização da linguagem, em seu ensino, na construção de bibliotecas de rotinas ou na participação no comitê de padronização, entre outras atividades. Algumas pessoas tiveram participação fundamental durante a história para o desenvolvimento.[30] Primeiramente, o próprio Bjarne Stroustrup, criador da linguagem e de seu primeiro compilador.[4] O cientista ainda participa na padronização e divulga o C++ no meio acadêmico. Andrew Koenig é outro pesquisador notável, bastante atuante na padronização e creditado pela técnica Koenig lookup (demonstrada em anexo). Já Scott Meyers é um doutor em ciência da computação, e escritor de diversos livros sobre o desenvolvimento de software utilizando a linguagem. Assim como Meyers, Herb Sutter é escritor de diversos livros sobre C++ e centenas de colunas e artigos, e um notável pesquisador sobre programação concorrente e multitarefa. Andrei Alexandrescu é considerado um dos maiores especialistas em programação C++ avançada. Na área de programação genérica destaca-se o programador russo Alexander Stepanov, a figura chave na criação da Standard Template Library.

Biblioteca Boost

Parte da comunidade de desenvolvimento do C++ foi responsável pela criação da Boost, um conjunto de bibliotecas que estendem a funcionalidade da linguagem, mais especificamente, da biblioteca padrão. O processo de disponibilização de bibliotecas da Boost é mais rápido do que o do comitê de padronização da linguagem, e o projeto acaba servindo como uma forma de teste em campo das bibliotecas, que eventualmente podem ser migradas para a especificação da linguagem. Para o TR1 do C++0x, pelo menos dez bibliotecas já foram aceitas.[31] A base de usuários da Boost é grande, o que reflete no controle de qualidade. Funcionalidades oferecidas vão desde bibliotecas gerais como smart_ptr[32] a abstrações do sistema operacional como o filesystem[33] a bibliotecas para usuários avançados como a MPL.[34]

Ferramentas

Ambientes de desenvolvimento

Abaixo é mostrada uma lista dos principais ambientes de desenvolvimento C++, sejam eles compiladores ou ambientes de desenvolvimento integrado (IDE).

Nome	Comentário	Tipo de licença	É compilador?	É IDE?	É depurador?	Plataformas
G++	Um componente do GCC, compilador padrão do Projecto GNU	Livre	Sim	Não	Sim	Unix, Linux, Mac OS X, Windows e AmigaOS
JetBrains CLion	IDE multiplataforma. Usa CMake como modelo de projeto e MinGW ou Cygwin como compiladores (em Windows) e GCC (em Linux).	Proprietário	Não	Sim	Não	Linux, Mac OS X e Windows
Dev-C++	IDE livre famosa entre iniciantes. Seu compilador é o MinGW, uma versão do G++ para Windows.	Livre	Não	Sim	Sim	Windows[nota 6]
Ultimate++	U++ é um framework C++ multi-plataforma, para desenvolvimento rápido de aplicações, focado na produtividade de programadores.	Livre	Não	Sim	Sim	Unix, Linux, Mac OS X e Windows
Intel C++	Produz código otimizado para processadores Intel	Proprietária	Sim	Não		Windows e Linux
Microsoft Visual C++	É o mais conhecido para a plataforma Windows, com ferramentas e tecnologias auxiliares para desenvolvimento nessa plataforma (como MFC, ATL, COM, entre outras). Oferece ainda uma versão gratuita com restrições de uso[35]	Proprietária	Sim	Sim	Sim	Windows
C++ Builder	Ferramenta da Embarcadero (anteriormente da Borland), que compartilha a mesma IDE do Delphi chamada de RAD Studio, sendo possível a utilização dos mesmos componentes visuais do Delphi (VCL) em um código C++.	Proprietária	Sim	Sim	Sim	Windows
Qt Creator	IDE especializada em (mas não restrita a) desenvolvimento na plataforma Qt. Utiliza o g++ como compilador e o gdb como depurador (ou seus equivalentes para MinGW em Windows).	Livre/Proprietária	Não	Sim	Não	Linux, Windows e Mac OS X
Open Watcom	Suporta plataformas antigas, até então sem suporte completo à biblioteca padrão	Livre	Sim	Sim		DOS, Windows, OS/2 e Netware
Comeau C++	Pode ser experimentado pela Internet[36]	Proprietária	Sim	Não		Windows, Linux e Solaris
Turbo C++	Possui versão gratuita[37] no sítio oficial e também uma versão paga.[38] É similar ao C++ Builder.	Proprietária	Sim	Sim		Windows
Eclipse	Disponível para C++ através da extensão CDT.[39]	Livre	Não	Sim	Sim	Windows, Linux, JVM
NetBeans	Possui versão especializada para C++ e também disponível através de um plugin.	Livre	Não	Sim	Sim	Windows, Linux, Mac OS X, JVM
Anjuta	Suporta muitas capacidades avançadas como gerenciamento de projetos e um poderoso editor de código fonte. Uma nova versão do Anjuta (Anjuta 2.*) que integra o Glade está em desenvolvimento ativo.	Livre	Não	Sim	Sim	Linux

Code::Blocks Ambiente aberto e multi-plataforma, em sua versão para Windows utiliza o compilador MinGW, apesar de também suportar outros compiladores como o Visual C++, Digital Mars, Borland C++ 5.5 e Open Watcom.[40] Livre Não Sim Sim Windows, Linux, Mac OS X

Digital Mars Proprietária Sim Sim Windows, DOS

Codelite Livre Sim Sim Windows, Linux, Mac OS X

Geany Livre Não Sim Não Windows, Linux

GNAT Programming Studio Utiliza o compilador GCC Livre Não Sim Sim Windows, Linux, Solaris

KDevelop Utiliza o compilador GCC Livre Não Sim Sim Windows, Linux

Arduino IDE Ambiente para compilação da placa de desenvolvimento/Prototipagem Arduíno (GCC) Livre Sim Sim Sim Windows, Linux, Mac OS, Android

Aplicativos desenvolvidos em C++

Abaixo segue uma lista de exemplos de aplicativos parcial ou totalmente escritos em C++, de acordo com Bjarne Stroustrup, que não garante sua precisão e veracidade, ainda que seja responsável por sua publicação.[41]

Adobe Acrobat

Adobe Illustrator

Adobe Photoshop

Amazon

BeOS

Blender

Common Desktop Environment

Doom III (motor de jogo)

eMule

Motores de busca Google, em especial Googlebot[42]

Série Half-Life

Internet Explorer

iPod (GUI)

KDE (Qt)

Lunar Magic

macOS

Maya

Mars Pathfinder, Opportunity e outras sondas da NASA

Microsoft Office

Microsoft Visual Studio (Visual Basic, Visual FoxPro, Visual C++)

Microsoft Windows (diversas versões)

Mozilla Firefox

Mozilla Thunderbird

MySQL

Máquina virtual Java

LibreOffice

Outlook Express

PCSX2 (Emulador de Playstation 2)

SETI@home

Symbian OS

Winamp

Tibia

Unreal Engine

Ver também

Wikilivros

O Wikilivros tem um livro chamado Programar em C++

Wikilivros

O Wikilivros tem um livro chamado Referência rápida de C++

Wikcionário

O Wikcionário tem o verbete C++.

Biblioteca padrão do C++

Biblioteca Boost

Operadores em C e C++

Orientação a objetos

Standard Template Library

Lista de exemplos de código C++

Lista de linguagens de programação

Um exemplo é o livro de Ivor Horton:

Ivor Horton (2004). Beginning ANSI C++ 3 ed. [S.l.]: APRESS. 827 páginas. ISBN 1-59059-227-1

O construtor padrão também é criado automaticamente pelo compilador caso o desenvolvedor não tenha definido nenhum outro construtor para a classe.

Dado um tipo de dado T, uma referência pode ser definida para nulo através de T& var = *(T*)0;. Notar entretanto que obter o conteúdo de nulo resulta em comportamento indefinido, o que indica má prática de programação.

Uma versão para Linux começou a ser desenvolvida, mas foi abandonada em 2002

Referências

«Overall Options - Using the GNU Compiler Collection (GCC)». gcc.gnu.org (em inglês). Consultado em 13 de dezembro de 2017

«Google C++ Style Guide» (em inglês). Google. Consultado em 13 de dezembro de 2017

«File Extension .C Details». filext.com (em inglês). Consultado em 13 de dezembro de 2017

Naomi Hamilton (25 de junho de 2008). «The A-Z of Programming Languages: C++». Entrevista com Bjarne Stroustrup (em inglês). Computerworld. Consultado em 25 de junho de 2008

«Timeline C++». Google Docs. Consultado em 14 de março de 2019

Bjarne Stroustrup (maio de 2005). «The Design of C++0x» (PDF) (em inglês). C/C++ Users Journal

«Doc No. 2697: ISO/IEC DTR 19768 (June 15, 2008) Minutes of WG21 Meeting June 8-15, 2008» (em inglês). Consultado em 26 de agosto de 2010

«We have an international standard: C++0x is unanimously approved» (em inglês). Consultado em 25 de novembro de 2011

Bjarne Stroustrup. «Quando o C++ foi inventado?» (em inglês). FAQ de Bjarne Stroustrup

Alexander Stepanov e Meng Lee (7 de março de 1994). «The Standard Template Library» (PDF) (em inglês). Hewlett-Packard, Technical Report X3J16/94-0095, WG21/N0482

«Descrição da norma» (em inglês)

Bjarne Stroustrup. «Where did the name "C++" come from?». FAQ de Bjarne Stroustrup. Consultado em 8 de março de 2007

Herb Sutter, Tom Plum (3 de março de 2003). «Why We Can't Afford Export» (PDF) (em inglês). Sítio oficial do comitê de padronização do C++. Consultado em 10 de setembro de 2007

«J16 Reunião 36/WG21, realizada entre 7 e 11 de abril de 2003» (em inglês)

«ISO/IEC 14882:2011» (em inglês). sítio ISO. 2 de setembro de 2011. Consultado em 25 de novembro de 2011

Stanley B. Lippman (1996). Inside the C++ Object Model. [S.l.]: Addison-Wesley Professional. 304 páginas. ISBN 978-0201834543

«Processo de compilação explicado pela IBM» (em inglês)

«Processo de compilação explicado pela HP» (em inglês)

«Programação C/C++ - Gabaritos - Templates». www.inf.pucrs.br. Consultado em 2 de março de 2019

«Classe template em C++ - Out4Mind». www.out4mind.com. Consultado em 2 de março de 2019

Marshall P. Cline; Greg Lomow; Mike Girou (1998). C++ FAQs 2 ed. [S.l.]: Addison-Wesley Professional. 624 páginas. ISBN 978-0201309836

Bjarne Stroustrup. «resource acquisition is initialization» (em inglês). Glossário de Bjarne Stroustrup

Bjorn Karlsson e Matthew Wilson (1 de outubro de 2004). «The Law of The Big Two» (em inglês). C++ Source. Consultado em 21 de abril de 2007

Herb Sutter realiza uma comparação entre as técnicas RAII e Dispose (que depende do conceito de finally) em seu blog pessoal:

Herb Sutter (31 de agosto de 2004). «C++ RAII compared with Java Dispose pattern» (em inglês). Página pessoal do autor. Consultado em 29 de setembro Verifique data em: |acessodata= (ajuda)

Bjarne Stroustrup. «Is C a subset of C++?» (em inglês). FAQ no sítio pessoal do autor. Consultado em 7 de fevereiro de 2008

David R. Tribble (5 de agosto de 2001). «Incompatibilities Between ISO C and ISO C++» (em inglês). Sítio pessoal do autor. Consultado em 28 de setembro de 2007

Andy (Março de 2001). «Parsing C++» (em inglês). Consultado em 7 de fevereiro de 2008

«The Java Language Specification LALR(1) Grammar». The Java Language Specification (em inglês). Sun Microsystems. 3 de abril de 1998. Consultado em 7 de fevereiro de 2008

«Boost.Spirit based C++ parser (library)» (em inglês). Grupo Boost. Consultado em 29 de abril de 2007

Scott Meyers (30 de agosto de 2006). «The Most Important C++ People…Ever» (em inglês). Artima Developer. Consultado em 30 de setembro de 2007

«Library Technical Report» (em inglês). Comitê de padronização do C++. 2 de julho de 2003. Consultado em 18 de outubro de 2008

«Boost: Smart Pointers» (em inglês). Sítio oficial da Boost. 25 de setembro de 2005. Consultado em 17 de fevereiro de 2008

«Boost: Boost Filesystem Library» (em inglês). Sítio oficial da Boost. 3 de junho de 2007. Consultado em 17 de fevereiro de 2008

«THE BOOST MPL LIBRARY» (em inglês). Sítio oficial da Boost. 15 de novembro de 2004. Consultado em 17 de fevereiro de 2008

«Visual Studio Express» (em inglês). MSDN, sítio da Microsoft Corporation. Consultado em 25 de outubro de 2007

«Test Drive Comeau C++ Online» (em inglês). Comeau Computing. 31 de março de 2007. Consultado em 7 de fevereiro de 2008

David Intersimone. «Antique Software: Turbo C++ version 1.01» (em inglês). Consultado em 24 de outubro de 2007

«Turbo C++» (em inglês). Borland. Consultado em 24 de outubro de 2007

«Eclipse C/C++ Development Tooling - CDT» (em inglês). Eclipse Foundation. Consultado em 24 de outubro de 2007

«Code::Blocks Features» (em inglês). Sítio oficial do Code::Blocks. Consultado em 24 de outubro de 2007

Bjarne Stroustrup. «C++ Applications» (em inglês). Sítio pessoal de Stroustrup. Consultado em 29 de março de 2018. I (Bjarne Stroustrup) don't make any guarantees about the accuracy of the list. I believe that it's accurate -- I trust the people who sent me examples, but I have not seen the source code myself.

«Descrição do Googlebot, desenvolvido em C++» (em inglês)

Bibliografia

Bjarne Stroustrup (1994). The Design and Evolution of C++. [S.l.]: Addison-Wesley. ISBN 0-201-54330-3

Andrei Alexandrescu e Herb Sutter (2004). C++ Design and Coding Standards: Rules and Guidelines for Writing Programs. [S.l.]: Addison-Wesley. ISBN 0-321-11358-6

CSS

Cascading Style Sheets (CSS) é um mecanismo para adicionar estilo (cores, fontes, espaçamento, etc.) a um documento web[1].

O código CSS pode ser aplicado diretamente nas tags ou ficar contido dentro das tags <style>. Também é possível, em vez de colocar a formatação dentro do documento, criar um link para um arquivo CSS que contém os estilos. Assim, quando se quiser alterar a aparência dos documentos vinculados a este arquivo CSS, basta modifica-lo.

Com a variação de atualizações dos navegadores, o suporte ao CSS pode variar. A interpretação dos navegadores pode ser avaliada com o teste Acid2, que se tornou uma forma base de revelar quão eficiente é o suporte de CSS, fazendo com que a nova versão em desenvolvimento do Firefox seja totalmente compatível a ele, assim como o Opera já é. O Doctype informado, ou a ausência dele, determina o quirks mode ou o strict mode, modificando o modo como o CSS é interpretado e a página desenhada.

Sintaxe

Resultado obtido no Acid2 satisfatoriamente.

CSS tem uma sintaxe simples, e utiliza uma série de palavras em inglês para especificar os nomes de diferentes propriedade de estilo de uma página.

Uma instrução CSS consiste em um seletor e um bloco de declaração. Cada declaração contém uma propriedade e um valor, separados por dois pontos (:). Cada declaração é separada por ponto e vírgula (;).[2]

Em CSS, seletores são usados para declarar a quais elementos de marcação um estilo se aplica, uma espécie de expressão correspondente. Os seletores podem ser aplicados a todos os elementos de um tipo específico, ou apenas aqueles elementos que correspondam a um determinado atributo; elementos podem ser combinados, dependendo de como eles são colocados em relação uns aos outros no código de marcação, ou como eles estão aninhados dentro do objeto de documento modelo.

Pseudoclasse é outra forma de especificação usada em CSS para identificar os elementos de marcação, e, em alguns casos, ações específicas de usuário para o qual um bloco de declaração especial se aplica. Um exemplo frequentemente utilizado é o :hover pseudoclasse que se aplica um estilo apenas quando o usuário 'aponta para' o elemento visível, normalmente, mantendo o cursor do mouse sobre ele. Isto é anexado a um seletor como em a:hover ou #elementid:hover. Outras pseudoclasses e pseudoelementos são, p. ex., :first-line, :visited ou :before. Uma pseudoclasse especial é :lang(c), "c".

Uma pseudoclasse seleciona elementos inteiros, tais como :link ou :visited, considerando que um pseudoelemento faz uma seleção que pode ser constituída por elementos parciais, tais como :first-line ou :first-letter.

Seletores podem ser combinados de outras formas também, especialmente em CSS 2.1, para alcançar uma maior especificidade e flexibilidade.[3]

Aqui está um exemplo que resume as regras acima:

```
selector [, selector2, ...][:pseudo-class] {
  property: value;
 [property2: value2;
```

...]

}

/* comment */

Seletores

Definição de estilo é um conjunto de propriedades visuais para um elemento, o CSS define regras que fazem as definições de estilo casarem com um elemento ou grupo de elemento, o documento pode conter um bloco de CSS num elemento style ou usando o elemento link apontando para um arquivo externo que contenha o bloco CSS.

Para uso com o CSS, foi criado o atributo class que todo elemento pode conter.

As regras de casamento para o CSS são chamadas de seletores, uma definição de estilo pode ser casada com um seletor ou um grupo de seletores separados por vírgula, um seletor pode casar um elemento por:

elemento do tipo : element_name { style definition; }

elemento do tipo com a classe : element_name.class_name { style definition; }

todos os elementos com a classe : .class_name { style definition; }

o elemento com o id : #id_of_element { style definition; }

casamento de um grupo : element_name_01, element_name_02, .class_name { style definition; }

Exemplos

p {text-align: right; color: #BA2;}

p.minhaclasse01 { color:#ABC; }

.minhaclasse02 { color:#CAD; }

iddomeuelemento { color:#ACD; }

p.minhaclasse03 .minhaclasse04 { color:#ACD; }

Exemplos

/* comentário em css, semelhante aos da linguagem c */

body

{

  font-family: Arial, Verdana, sans-serif;

  background-color: #FFF;

```
  margin: 5px 10px;
}
```

O código acima define fonte padrão Arial, caso não exista, substitui por Verdana, caso não exista, define qualquer fonte sans-serif. Define também a cor de fundo do corpo da página.

Sua necessidade adveio do fato do HTML, aos poucos, ter deixado de ser usado apenas para criação de conteúdo na web, e portanto havia uma mistura de formatação e conteúdo textual dentro do código de uma mesma página. Contudo, na criação de um grande portal, fica quase impossível manter uma identidade visual, bem como a produtividade do desenvolvedor. É nesse ponto que entra o CSS.

As especificações do CSS podem ser obtidas no site da W3C "World Wide Web Consortium", um consórcio de diversas empresas que buscam estabelecer padrões para a Internet.

É importante notar que nenhum navegador suporta igualmente as definições do CSS. Desta forma, o web designer deve sempre testar suas folhas de estilo em navegadores de vários fabricantes, e preferencialmente em mais de uma versão, para se certificar de que o que foi codificado realmente seja apresentado da forma desejada.

Exemplo de CSS aplicado em XML

O arquivo de exemplo XML renderizado no Mozilla Firefox.

Arquivo *.XML com ligação para uma folha de estilos em cascata:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="css.css"?>
<schedule>
  <date>Tuesday 20 June</date>
  <programme>
    <starts>6:00</starts>
    <title>News</title>
    With Michael Smith and Fiona Tolstoy.
    Followed by Weather with Malcolm Stott.
  </programme>
```

```
  <programme>
    <starts>6:30</starts>
    <title>Regional news update</title>
    Local news for your area.
  </programme>
  <programme>
    <starts>7:00</starts>
    <title>Unlikely suspect</title>
    Whimsical romantic crime drama starring Janet
    Hawthorne and Percy Trumpp.
  </programme>
</schedule>
```

O arquivo *.CSS que formata o XML anterior:

```
@media screen {
  schedule {
    display: block;
    margin: 10px;
    width: 300px;
  }
  date {
    display: block;
    padding: 0.3em;
    font: bold x-large sans-serif;
    color: white;
    background-color: #C6C;
  }
  programme {
    display: block;
    font: normal medium sans-serif;
  }
```

```
   programme > * { /* All children of programme elements */
     font-weight: bold;
     font-size: large;
   }
   title {
     font-style: italic;
   }
 }
```

CSS e JavaScript

Pop-up não-bloqueável

```
<html>
  <head>

   <style type="text/css">
# popup {
       position: absolute;
       top: 30%;
       left: 30%;
       width: 300px;
       height: 150px;
       padding: 20px 20px 20px 20px;
       border-width: 2px;
       border-style: solid;
       background: #ffffa0;
       display: none;
     }
   </style>

   <script type="text/javascript">
     function abrir() {
       document.getElementById('popup').style.display = 'block';
```

```
      setTimeout ("fechar()", 3000);
    }

    function fechar() {
      document.getElementById('popup').style.display = 'none';
    }
  </script>
 </head>
 <body onload="abrir()" >
  <div id="popup" class="popup">
    Esse é um exemplo de popup utilizando o elemento <code>div</code>. Dessa maneira esse
    pop-up não será bloqueado.

    <small><a href="javascript: fechar();">[X]</a></small>
  </div>

<a href="javascript: abrir();">Abrir pop-up</a>

<a href="javascript: fechar();">Fechar pop-up</a>
 </body>
</html>
```

Referências

«Cascading Style Sheets». www.w3.org (em inglês). Consultado em 8 de setembro de 2020

«W3C CSS2.1 specification for rule sets, declaration blocks, and selectors». W3.org. Consultado em 8 de setembro de 2020

Ver a definição completa de seletores no website W3C.

HTML

HTML (abreviação para a expressão inglesa HyperText Markup Language, que significa Linguagem de Marcação de Hipertexto) é uma linguagem de marcação utilizada na construção

de páginas na Web. Documentos HTML podem ser interpretados por navegadores. A tecnologia é fruto da junção entre os padrões HyTime e SGML.

HyTime é um padrão para a representação estruturada de hipermídia e conteúdo baseado em tempo. Um documento é visto como um conjunto de eventos concorrentes dependentes de tempo (como áudio, vídeo, etc.), conectados por hiperligações (em inglês: hyperlink e link). O padrão é independente de outros padrões de processamento de texto em geral.

SGML é um padrão de formatação de textos. Não foi desenvolvido para hipertexto, mas tornou-se conveniente para transformar documentos em hiper-objetos e para descrever as ligações.

História

Tim Berners-Lee em 2008.

Tim Berners-Lee (físico britânico) criou o HTML original (e outros protocolos associados como o HTTP) em uma estação NeXTcube usando o ambiente de desenvolvimento NeXTSTEP. Na época a linguagem não era uma especificação, mas uma coleção de ferramentas para resolver um problema de Tim: a comunicação e disseminação das pesquisas entre ele e seu grupo de colegas. Sua solução, combinada com a então emergente internet pública (que tornar-se-ia a Internet) ganhou atenção mundial.

As primeiras versões do HTML foram definidas com regras sintáticas flexíveis, o que ajudou aqueles sem familiaridade com a publicação na Web. Através do tempo, a utilização de ferramentas para autoria de HTML aumentou, assim como a tendência em tornar a sintaxe cada vez mais rígida. Apesar disso, por questões históricas (retrocompatibilidade), os navegadores ainda hoje conseguem interpretar páginas web que estão longe de ser um código HTML válido.

A linguagem foi definida em especificações formais na década de 1990, inspiradas nas propostas originais de Tim Berners-Lee em criar uma linguagem baseada em SGML para a Internet. A primeira publicação foi esboçada por Berners-Lee e Dan Connolly, e publicada em 1993 na IETF como uma aplicação formal para o SGML (com uma DTD em SGML definindo a gramática). A IETF criou um grupo de trabalho para o HTML no ano seguinte, e publicou o HTML 2.0 em 1995. Desde 1996, as especificações HTML vêm sendo mantidas, com o auxílio de fabricantes de software, pelo World Wide Web Consortium (W3C).[1] Apesar disso, em 2000 a linguagem tornou-se também uma norma internacional (ISO/IEC 15445:2000). A recomendação HTML 4.01 foi publicada no final de 1999 pelo W3C. Uma errata ainda foi lançada em 2001.

Desde a publicação do HTML 3.5 no final de 1997, o grupo de trabalho da W3C tem cada vez mais — e de 2002 a 2006, de forma exclusiva — focado no desenvolvimento do XHTML, uma especificação HTML baseada em XML que é considerada pela W3C como um sucessor do HTML.[2][3][4] O XHTML faz uso de uma sintaxe mais rigorosa e menos ambígua para tornar o HTML mais simples de ser processado e estendido.

Em janeiro de 2008 o W3C publicou a especificação do HTML5 como Working Draft. Apesar de sua sintaxe ser semelhante a de SGML, o HTML5 abandonou qualquer tentativa de ser uma aplicação SGML e, definiu explicitamente sua própria serialização "html", além de uma alternativa baseada em XML, o XHTML5.[5]

Várias versões HTML foram publicadas:[6]

Versão Ano

HTML 1991

HTML 2.0 1995

HTML 3.2 1997

HTML 4.01 1999

XHTML 2000

HTML5 2014

HTML5.1 2016 e 2017 (2ª ed.)

HTML5.2 2017

Marcas (tags)

Tabela de cores

Todo documento HTML possui marcadores (do inglês: tags), palavras entre parênteses angulares (chevron) (< e >); esses marcadores são os comandos de formatação da linguagem. Um elemento é formado por um nome de marcador (tag), atributos, valores e filhos (que podem ser outros elementos ou texto). Os atributos modificam os resultados padrões dos elementos e os valores caracterizam essa mudança. Exemplo de um elemento simples (não possui filhos):

```
<hr />
```

Exemplo de um elemento composto (possui filhos):

```
<a href="http://pt.wikipedia.org/">Wikipédia</a>
```

<a> é o marcador de abertura

 é o marcador de fechamento

href é o atributo onde é definido a url, que será acessada ao clicar no link.

Outro exemplo de elemento composto (possui filhos):

 Wikipédia

p = marcador que define um parágrafo.

a = marcador que define uma hiperligação.

href = atributo que define a url da hiperligação.

target = atributo que define a forma como a hiperligação será aberta.

_self = valor do atributo Target que define que a hiperligação será aberta na mesma guia.

/ = define o fechamento do elemento

Isso é necessário porque os marcadores servem para definir a formatação de uma porção do documento, e assim marcamos onde começa e termina o conteúdo que receberá a formatação ou marcação necessária, específica. Alguns elementos são chamados “vazios”, pois não marcam uma região de texto, apenas inserem algum elemento no documento.

Cada elemento tem os seus atributos possíveis e os seus valores. Um exemplo, é o atributo href que pode ser usado com o elemento a, com o link mas que não pode ser usado com o elemento meta. Isso quer dizer que devemos saber exatamente quais os atributos e valores possíveis para cada elemento.

De uma maneira geral o HTML é um poderoso recurso, sendo uma linguagem de marcação muito simples e acessível voltada para a produção e compartilhamento de documentos, imagens, vídeos e áudio via streaming.

Na sua versão mais recente, o HTML5, é possível criar marcadores personalizados com JavaScript, linguagem de programação diretamente compatível com o HTML. Cada tag pode ter uma função específica utilizando uma API (Interface de programação de aplicações) diferente, assim como seus nomes e estilos.

Edição de documentos HTML

Os documentos em HTML são arquivos de texto simples que podem ser criados e editados em qualquer editor de textos comum, como o Bloco de Notas do Windows, ou o TextEdit, do Macintosh. Para facilitar a produção de documentos, no mercado existem editores HTML específicos, com recursos sofisticados, que facilitam a realização de tarefas repetitivas,

inserção de objetos, elaboração de tabelas e outros recursos (Ver lista abaixo). Basicamente dividem-se em dois tipos:

Editores de texto fonte: inserem automaticamente os marcadores, orientando a inserção de atributos e marcações

Editores WYSIWYG: oferecem ambiente de edição com um "esboço" resultado final das marcações

Estrutura básica de um documento

A estrutura básica de um documento HTML (Hyper Text Markup Language - Linguagem de Marcação de Hypertexto), apresenta as seguintes marcações:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="description" content="a descrição do seu site em no máximo 90 caracteres">
    <meta name="keywords" content="escreva palavras-chaves curtas, máximo 150 caracteres">
    <title>Título do Documento</title>
  </head>
  <body>
  <!-- Aqui fica a página que será visível para todos, onde pode-se inserir
  textos, imagens, links para outras páginas, etc, geralmente usa-se: -->

  <div>Tag para criar-se uma 'caixa', um bloco, mais utilizada com "Cascading Style Sheets
    (Folhas de Estilo em Cascata)</div>

  <span>Tag para modificação de uma parte do texto da página</span>

  <img src="endereço_de_uma_imagem.jpg" />

  <a href="http://www.wikipedia.org">Wikipedia, A Enciclopédia Livre</a>
```

</body>

</html>

Os marcadores HTML não são sensíveis à caixa, portanto tanto faz escrever <HTML>, <Html>, <html> ou <HtMl>.

Os marcadores básicos de HTML, cuja presença é altamente recomendada nas páginas são:

<html>: define o início de um documento HTML e indica ao navegador que todo conteúdo posterior deve ser tratado como uma série de códigos HTML

<head>: define o cabeçalho de um documento HTML, que traz informações sobre o documento que está sendo aberto

<body>: define o conteúdo principal, o corpo do documento. Esta é a parte do documento HTML que é exibida no navegador. No corpo podem-se definir atributos comuns a toda a página, como cor de fundo, margens, e outras formatações.

Cabeçalho

Dentro do cabeçalho podemos encontrar os seguintes elementos:

<title>: define o título da página, que é exibido na barra de título dos navegadores

<style type="text/css">: define formatação em CSS

<script type="text/javascript">: define programação de certas funções em página com scripts, podendo adicionar funções de JavaScript

<link>: define ligações da página com outros arquivos como feeds, CSS, scripts, etc

<meta>: define propriedades da página, como codificação de caracteres, descrição da página, autor, etc

São meta informações sobre documento. Tais campos são muito usados por mecanismos de busca (como o Google, Yahoo!, Bing) para obterem mais informações sobre o documento, a fim de classificá-lo melhor. Por exemplo, pode-se adicionar o código <meta name="description" content="descrição da sua página" /> no documento HTML para indicar ao motor de busca que texto de descrição apresentar junto com a ligação para o documento. Para o motor de busca Google, por exemplo, elementos meta como keywords não são utilizadas para indexar páginas. Apenas <title> e a meta <description> são usadas para descrever a página indexada.[7]

Obs: Os marcadores <style> e <script> servem tanto para delimitar os espaços usados pelos códigos na página quanto para invocar códigos existentes em outros arquivos externos.

Corpo

Trecho de código HTML.

Dentro do corpo podemos encontrar outros vários marcadores que irão moldar a página, como por exemplo:

<h1>, <h2>, ... <h6>: Títulos que variam de tamanho dentro das prioridades (sua aparência pode ser alterada com CSS - Cascade Style Sheet - Folhas de Estilo em Cascata).

<p>: Parágrafo.

: quebra de linha.

<table>: cria uma tabela (linhas são criadas com <TR> e novas células com <TD>, já os cabeçalhos das colunas são criados com os marcadores <THead><TH> e os rodapés com <TFooter><TR><TD>).

: determina uma divisão na página a qual pode possuir variadas formatações.

<b>, <i>, <u> e <s>: negrito, itálico, sublinhado e riscado, respectivamente.

<img />: imagem.

<a>: hiper-ligação para um outro local, seja uma página, um e-mail ou outro serviço.

<textarea>: caixa de texto (com mais de uma linha); estas caixas de texto são muito usadas em blogs, elas podem ser auto selecionáveis e conter outros códigos a serem distribuídos.

<abbr>: abreviação (sigla simplesmente abreviada).

<cite>: citação.

<address>:Endereço.

Cores

As cores devem ser declaradas em CSS, com o atributo style, que funciona em diversos elementos, como por exemplo:

Texto

Onde COR pode ser o nome da cor em inglês, em decimal, hexadecimal, RGB, RGBA ou HSLA. Exemplos: Tabela de cores e Lista de cores.

Hiperligações

Uma possibilidade importante dos documentos HTML é a de fazer hiperligações. Para isso usa-se o marcador <a> (do inglês, anchor). Esta tem os atributos: href que define o alvo da hiperligação (que pode ser uma página de Internet, uma parte da mesma página ou um

endereço de email) ou name que define um alvo nessa página (a onde se pode fazer uma hiperligação usando o marcador a com o atributo href). Exemplos:

Clique aqui para aceder à página principal da Wikipédia em português.

texto

Em que nome e texto podem ser substituídos por o que se desejar. Depois usa-se   para hiperligar a este "anchor".

Diferença entre target="_blank" e target="_new"

target="_blank" é usado para abrir links em várias janelas e target="_new" ou target="booger" é usado para abrir vários links em uma janela.[8]

Exemplos

Título

Título

Título

Página em branco é usado about:blank na url do link.

Exemplos:

Página em branco

Página em branco

Página em branco

Caracteres especiais e símbolos

Os caracteres especiais definem-se usando comandos que começam com & e terminam com um ;. Alguns exemplos incluem á (á), à (à), ã (ã), â (â), ä (ä) e ç (ç). Qualquer outra vogal pode ser substituída pelo a destes exemplos, incluindo maiúsculas.

Referências

Dave Raggett (1998). Raggett on HTML 4. [S.l.]: Addison-Wesley's. pp. chap. 2: A history of HTML. ISBN 0-201-17805-2

«HTML working group charter (2000–2002)». World Wide Web Consortium

«HTML working group charter (2002–2004)». World Wide Web Consortium

«HTML Working Group Roadmap». World Wide Web Consortium

Karl Dubost (15 de janeiro de 2008). «HTML5, one vocabulary, two serializations» (em inglês). W3C. Consultado em 29 de outubro de 2011

«Introdução ao HTML». www.escolaw3.com. Consultado em 25 de março de 2017

«Official Google Webmaster Central Blog: Google does not use the keywords meta tag in web ranking». googlewebmastercentral.blogspot.com. Consultado em 21 de fevereiro de 2011

target="_blank" vs. target="_new"(em inglês)