Instalando e criando sua primeira aplicação com o Python

IDE´s

A sigla IDE (Integrated Development Environment) trata-se de um programa que reúne uma série de ferramentas que facilitam a vida do programador. Dentre seus recursos, podemos citar a presença:

* Compilador: onde você rodará seu programa);
* Editor: onde você escreverá seu código;
* Depurador (debugger): para entender por que o seu código não está funcionando.

Além desses recursos fundamentais, muitas IDEs permitem personalização do seu ambiente de trabalho, customizando desde sua aparência, com temas e cores, ou até alterar totalmente seu código fonte. Tudo para deixar sua vida mais fácil. Sem mais delongas, vamos para a lista!

Anaconda - Spyder

Escolha Anaconda se você:

* É novo no conda ou Python
* Quer ter o Python e mais de 150 pacotes científicos instalados automaticamente de uma só vez
* Tenha tempo e espaço em disco (alguns minutos e 3 GB) e/ou
* Não deseja instalar cada um dos pacotes que deseja usar individualmente.

Escolha Miniconda se você:

* Não se importe de instalar cada um dos pacotes que você deseja usar individualmente.
* Não há tempo ou espaço em disco para instalar mais de 150 pacotes de uma só vez e/ou
* Apenas deseja acesso rápido ao Python e aos comandos conda e deseje resolver os outros programas posteriormente.

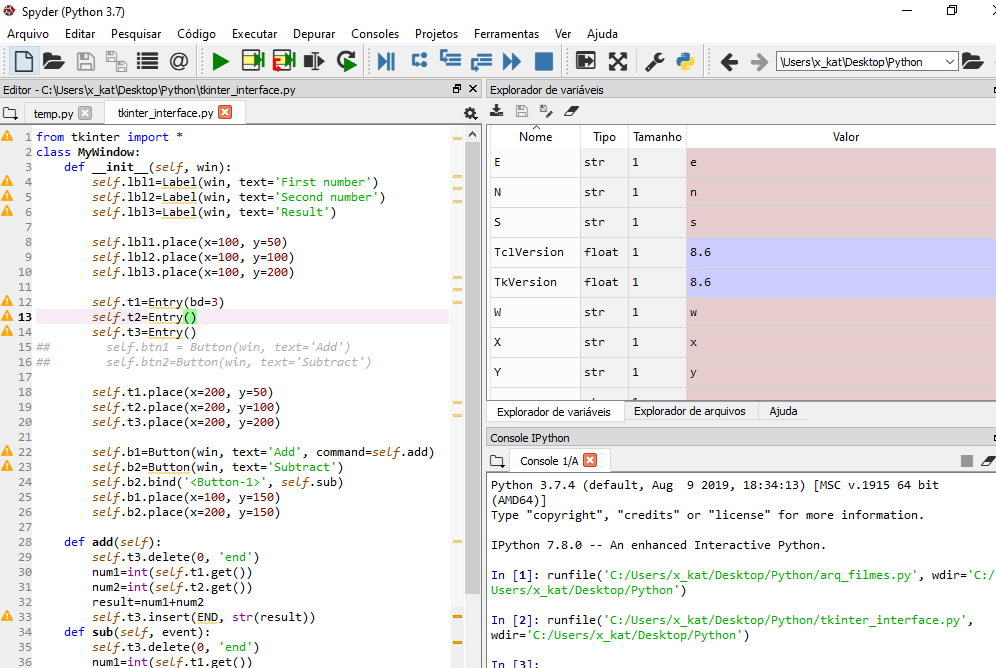
Observe que Conda é o gerenciador de pacotes (por exemplo, o comando **conda list** exibe todos os pacotes instalados no ambiente), enquanto Anaconda e Miniconda são distribuições. Uma distribuição de software é uma coleção de pacotes, pré-criados e pré-configurados, que podem ser instalados e usados ​​em um sistema. Um gerenciador de pacotes é uma ferramenta que automatiza o processo de instalação, atualização e remoção de pacotes.

O Anaconda é uma distribuição completa do software central no ecossistema PyData e inclui o próprio Python junto com os binários de várias centenas de projetos de código aberto de terceiros. O Miniconda é essencialmente um instalador de um ambiente conda vazio, contendo apenas Conda, suas dependências e Python.

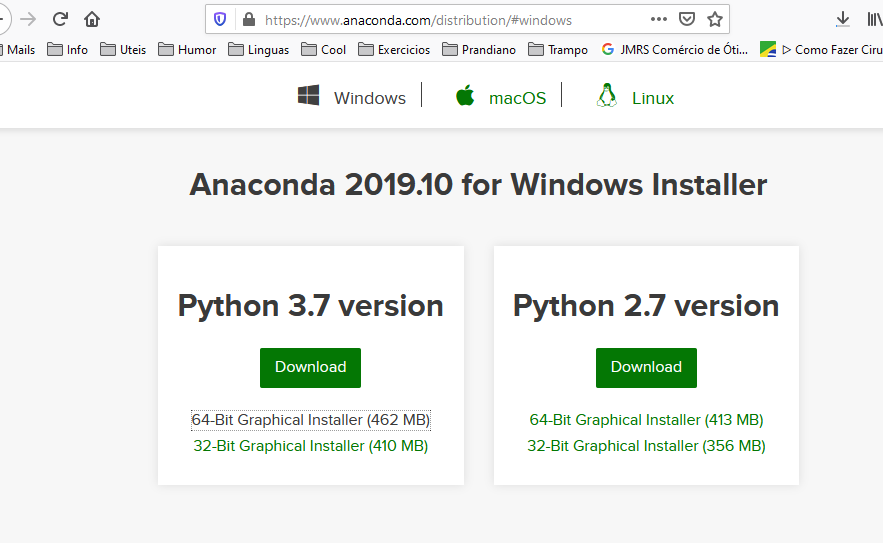
Depois que o Conda estiver instalado, você poderá instalar o pacote que precisar do zero, juntamente com qualquer versão desejada do Python.

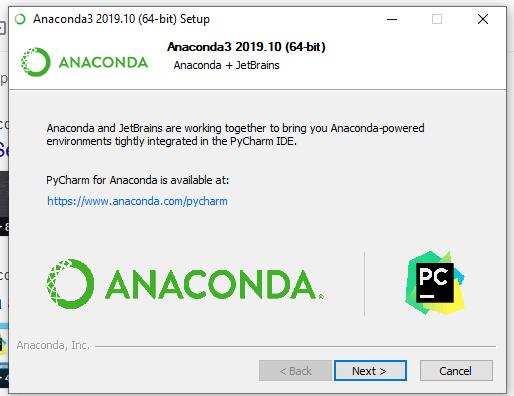
Se você está aprendendo Data Science e optou por instalar o pacote do Anaconda, que contém além do Python em si, muitas de suas bibliotecas, com certeza já ouviu falar do Spyder.

O Spyder não disponibiliza uma gama de personalização, mas tem a vantagem de ser uma ferramenta leve e simples. Ele conta com um compilador e depurador bem prático, com cores que salientam bem os erros, funções e variáveis do código. Destaque para o explorador de variáveis, que mostra o valor armazenado dentro de cada uma, dispensando comandos print na tela.

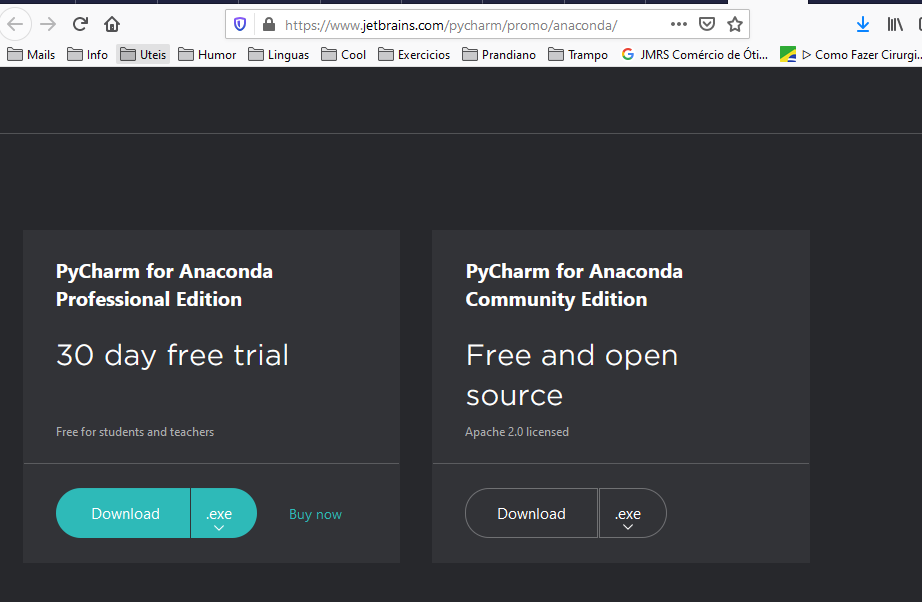


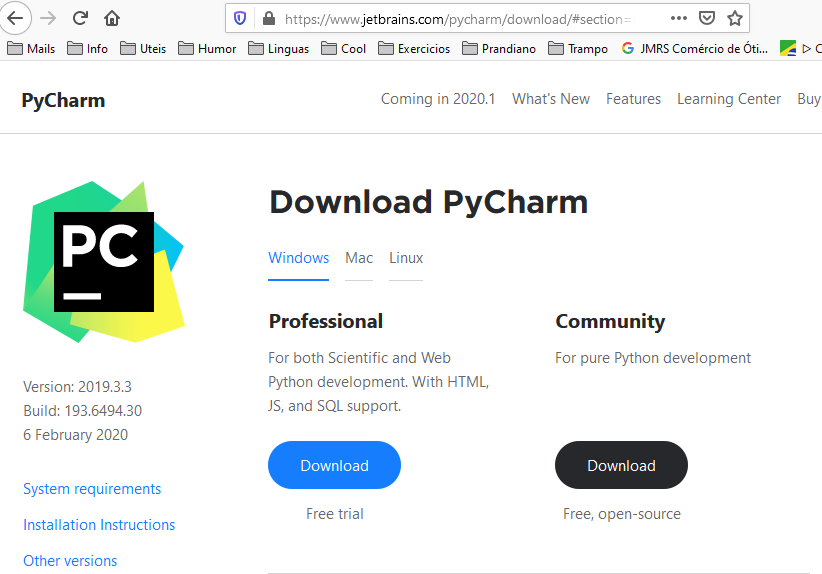
https://www.anaconda.com/distribution/#windows





https://www.jetbrains.com/pycharm/promo/anaconda/

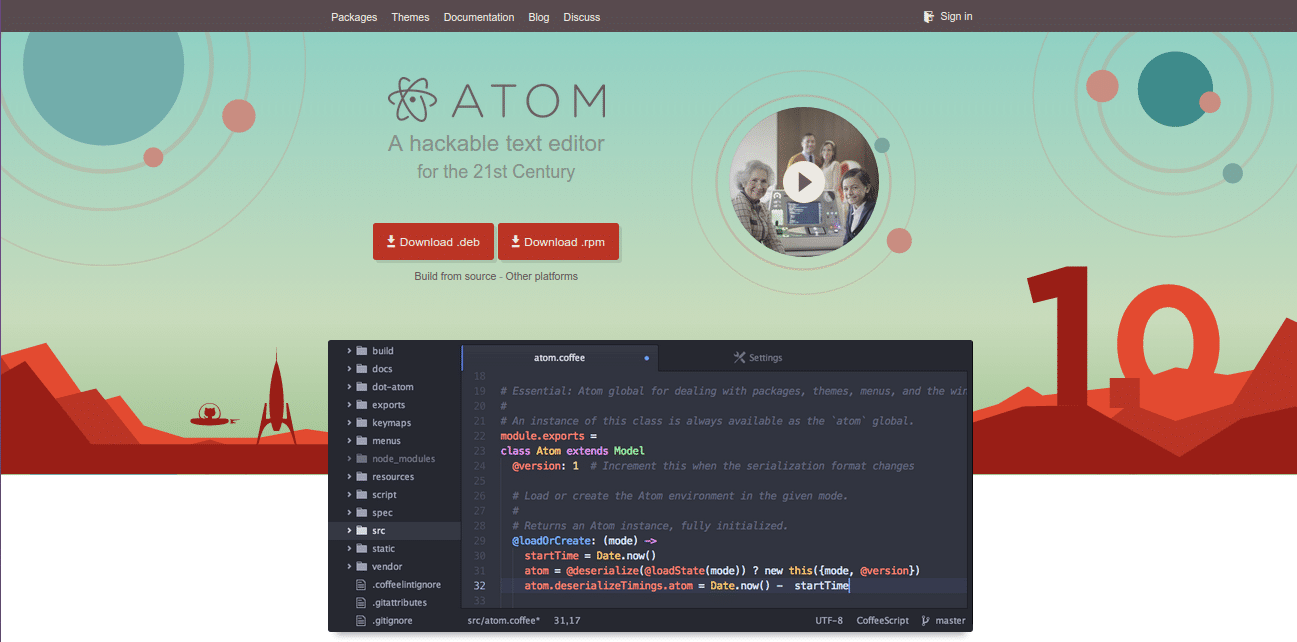


https://www.jetbrains.com/pycharm/download/#section=windows

PyCharm

O PyCharm, além de uma interface muito limpa e personalizável, é ideal para aqueles que estão dando os primeiros passos com Python. Com diversas funções para customizar o workspace, o PyCharm conta com funções internas de Python para que você possa acessar rapidamente e sem erros.

Atom



O Atom, criado pela equipe do Github, permite a modificação do código-fonte permitindo customizar o workspace com um aumento de produtividade. Tem uma comunidade muito ativa que estão lançando pacotes para melhorar sua experiência da programação. Como desvantagem, seria seu desempenho, caso seu computador não for tão potente.

Visual Studio Code

O Visual Studio Code possui integração nativa com o Github, o que permite fazer commits e sincronizações direto na tela (no Atom, é necessário a instalação de pacotes). A customização da interface com temas e cores também está presente nessa ferramenta e, assim como no Spyder, ele possui um explorador de variáveis, que facilita a visualização dos valores e tipos de dados armazenados nelas. Conta também com depuradores e compiladores nativos.

Programação funcional

O paradigma funcional foi inspirado em um modelo de trabalho onde eu não dito ao meu código, passo a passo, o que ele deve fazer, quando e como. Linguagens puramente funcional é um código composto de múltiplas funções que se compõem em momento oportuno para resolver um problema. Abaixo um resumo cronológico de eventos importantes eventos associados a origem da programação funcional.

|  |  |  |  |
| --- | --- | --- | --- |
| **Nome** | **Criador** | **Ano** | **Origem** |
| Axiomas de Peano | Por Richard Dedekind  Por Giuseppe Peano | Em 1888(Dedekind)  Em 1889(Peano) | Braunschweig, Alemanha  Universidade de Turim< Italia |
| Entscheidungsproblem | Por David Hilbert | Em 1900 e 1928 | Congresso Internacional de Matemáticos em Paris |
| Funções recursivas | Por Kurt Gödel e Jacques Herbrand | Em 1934 | Instituto de Estudos Avançados de Princeton |
| Máquinas de Turing | Alan Turing | Em 1936(idéias)  Em 1937(publicação) | Londres |
| Funções lambda-definíveis | Por Alonzo Church e Stephen Kleene | Em 1936 | [Hudson, Ohio](https://pt.wikipedia.org/wiki/Hudson_(Ohio)) |
| LISP, List Processing | Por [John McCarthy](https://pt.wikipedia.org/wiki/John_McCarthy) | Em [1956](https://pt.wikipedia.org/wiki/1958)(idéias em IA),  Em [1958](https://pt.wikipedia.org/wiki/1958)(implementação) | MIT |
| ML - Standard ML | por [Robin Milner](https://pt.wikipedia.org/wiki/Robin_Milner) e outros | no [anos 1970](https://pt.wikipedia.org/wiki/Anos_1970)(ML)  no [anos 1980](https://pt.wikipedia.org/wiki/Anos_1970)(Standard) | [Universidade de Edimburgo](https://pt.wikipedia.org/wiki/Universidade_de_Edimburgo), [Reino Unido](https://pt.wikipedia.org/wiki/Reino_Unido) |
| Scheme | Por Guy L. Steele e Gerald Jay Sussman | Em 1975 | Inteligência Artificial e Ciência da Computação do MIT |
| Haskell | FPCA - Functional Programming Languages and Computer Architecture | Em [1990](https://pt.wikipedia.org/wiki/1990) | no Oregon |
| OCaml - Objective Caml | [INRIA](https://pt.wikipedia.org/wiki/Institut_National_de_Recherche_en_Informatique_et_en_Automatique) e pela [École Normale Supérieure](https://pt.wikipedia.org/wiki/%C3%89cole_Normale_Sup%C3%A9rieure) | Em [199](https://pt.wikipedia.org/wiki/1996)0 (origem)  Em [1996](https://pt.wikipedia.org/wiki/1996) (versão POO) | em Paris |
| F# | Don Syme | Em 2005 | na [Microsoft Research](https://pt.wikipedia.org/wiki/Microsoft_Research) |

Axiomas de Peano

Bem antes do século XIX, os matemáticos usavam o princípio de definir uma função por indução, mas, somente com a necessidade de formalizar a aritmética na década de 1860 que Hermann Grassmann mostrou que muitos fatos em aritmética poderiam ser derivados de fatos mais básicos sobre a operação e indução sucessivas. Em 1881, Charles Sanders Peirce fornece uma axiomatização da aritmética de números naturais. Em 1888, Richard Dedekind propôs outra axiomatização da aritmética de números naturais e provou, usando axiomas aceitos, que tal definição define uma função única, e ele a aplicou à definição das funções m + n, mxn, e M. Com base neste trabalho, em 1889, Giuseppe Peano publicou uma versão simplificada deles como uma coleção de nove axiomas para números inteiros positivos surgindo os Axiomas de Peano, também conhecidos na teoria dos números como postulados de Peano. Posteriormente, em 1891, Peano reconhece que seus axiomas são, na verdae, de Dedekind.

Como os axiomas para geometria criados pelo matemático grego Euclides (c. 300 aC), os axiomas de Peano foram criados para fornecer uma base rigorosa para os números naturais (0, 1, 2, 3, ...) usados ​​em aritmética, teoria dos números e teoria de conjuntos. Em particular, os axiomas Peano permitem que um conjunto infinito seja gerado por um conjunto finito de símbolos e regras.

Os axiomas Peano contêm três tipos de declarações. O primeiro axioma afirma a existência de pelo menos um membro do conjunto de números naturais.

|  |  |
| --- | --- |
| **Axioma** | **Declaração 1** |
| 1 | a constante 0 é um número natural. |

A formulação original de Peano dos axiomas usava 1 em vez de 0 como o "primeiro" número natural. Essa escolha é arbitrária, pois o axioma 1 não dota a constante 0 com nenhuma propriedade adicional. No entanto, como 0 é a identidade aditiva na aritmética, as formulações mais modernas dos axiomas de Peano começam em 0.

Os próximos quatro (2 a 5) são afirmações gerais sobre igualdade; em tratamentos modernos, eles geralmente não são tomados como parte dos axiomas de Peano, mas como axiomas da "lógica subjacente".

|  |  |
| --- | --- |
| **Axioma** | **Lógica Subjacente** |
| 2 | Para todo número natural x, x = x. Ou seja, a igualdade é reflexiva. |
| 3 | Para todos os números naturais x e y, se x = y, então y = x. Ou seja, a igualdade é simétrica. |
| 4 | Para todos os números naturais x, ye z, se x = ye y = z, então x = z. Ou seja, a igualdade é transitiva. |
| 5 | Para todos a e b, se b é um número natural e a = b, então a também é um número natural. Ou seja, os números naturais são fechados sob igualdade. |

Os próximos três axiomas são declarações de primeira ordem sobre números naturais que expressam as propriedades aritméticas fundamentais da operação sucessora, geralmente representados como um conjunto N. Os símbolos não lógicos para os axiomas consistem em um símbolo constante 0 e um símbolo de função "sucessora" de valor único S (unária).

|  |  |
| --- | --- |
| **Axioma** | **Declaração de primeira ordem** |
| 6 | Para todo número natural n, S (n) é um número natural. Ou seja, os números naturais são fechados em S. Todo número natural tem um sucessor nos números naturais. |
| 7 | Para todos os números naturais m e n, m = n se e somente se S (m) = S (n). Ou seja, S é uma injeção. Se o sucessor de dois números naturais for o mesmo, os dois números originais serão os mesmos. |
| 8 | Para todo número natural n, S (n) = 0 é falso. Ou seja, não existe um número natural cujo sucessor seja 0. Zero não é o sucessor de nenhum número natural. |

Juntamente com o axiomas 1, eles definem uma representação unária da noção intuitiva de números naturais: o número 1 pode ser definido como S(0), 2 como S(S(0)) etc. No entanto, considerando a noção de números naturais como sendo definida por esses axiomas, os axiomas não implicam que a função sucessora gere toda a números diferentes de 0. Em outras palavras, eles não garantem que todo número natural que não seja zero tenha sucesso em algum outro número natural. A noção intuitiva de que cada número natural pode ser obtido aplicando sucessivamente a zero o suficiente requer um axioma adicional, que às vezes é chamado axioma da indução.

|  |  |
| --- | --- |
| **Axioma** | **Declaração de segunda ordem** |
| 9 | Se K é um conjunto tal que: 0 está em K e para todo número natural n, n estando em K implica que S (n) está em K, então K contém todos os números naturais.  Às vezes, o axioma de indução é declarado da seguinte forma:  Se φ é um predicado unário tal que: φ (0) é verdadeiro e para todo número natural n, φ (n) ser verdadeiro implica que φ (S (n)) é verdadeiro, então φ (n) é verdadeiro para todo número natural n.  Ou seja, Se um conjunto contiver zero e o sucessor de cada número estiver no conjunto, o conjunto conterá os números naturais. |

O princípio da indução, pode ser usado para estabelecer propriedades para um número infinito de casos sem precisar fornecer um número infinito de provas. Em particular, dado que P é uma propriedade e zero tem P e que sempre que um número natural tem P, seu sucessor também possui P, segue-se que todos os números naturais têm P.

Na formulação original de Peano, o axioma de indução é um axioma de segunda ordem. Agora é comum substituir esse princípio de segunda ordem por um esquema de indução de primeira ordem mais fraco. Existem diferenças importantes entre as formulações de segunda e primeira ordem que não serão exploradas neste estudo.

## Entscheidungsproblem - o Problema de decisão

Os Problemas de Hilbert são uma lista de 23 problemas em matemática propostos pelo matemático alemão David Hilbert(1862 - 1943) na conferência do Congresso Internacional de Matemáticos de Paris em 1900. Nenhum dos problemas havia tido solução até então, e vários deles acabaram se tornando muito influentes na matemática do século XX. Nessa conferência, ele publicou 10 dos problemas (1, 2, 6, 7, 8, 13, 16, 19, 21, e 22). O resto da lista foi publicado mais tarde.

|  |  |  |
| --- | --- | --- |
| **Problema** | **Situação** | **Enunciado** |
| [1](https://pt.wikipedia.org/wiki/Primeiro_problema_de_Hilbert) | Resolvido[1](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | Provar a [hipótese do continuum](https://pt.wikipedia.org/wiki/Hip%C3%B3tese_do_continuum) (HC) de [Cantor](https://pt.wikipedia.org/wiki/Georg_Cantor) |
| [2](https://pt.wikipedia.org/wiki/Segundo_problema_de_Hilbert) | Resolvido[2](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | Demonstrar a consistência dos axiomas da [aritmética](https://pt.wikipedia.org/wiki/Aritm%C3%A9tica) |
| [3](https://pt.wikipedia.org/wiki/Terceiro_Problema_de_Hilbert) | Resolvido[3](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | Pode-se provar que dois [tetraedros](https://pt.wikipedia.org/wiki/Tetraedro) têm o mesmo [volume](https://pt.wikipedia.org/wiki/Volume) (sob certas condições)? |
| [4](https://pt.wikipedia.org/wiki/Quarto_problema_de_Hilbert) | Vago Demais[4](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | Construir todos os espaços métricos em que as linhas são [geodésicas](https://pt.wikipedia.org/wiki/Geod%C3%A9sia) |
| [5](https://pt.wikipedia.org/wiki/Quinto_problema_de_Hilbert) | Resolvido[5](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | Todo [grupo](https://pt.wikipedia.org/wiki/Grupo_(matem%C3%A1tica)) contínuo é automaticamente um grupo diferencial? |
| [6](https://pt.wikipedia.org/wiki/Sexto_problema_de_Hilbert) | Não-matemático[6](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | Transformar toda a Física em [axiomas](https://pt.wikipedia.org/wiki/Axiomas) |
| 7 | Resolvido[7](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | a b é transcendente para a ≠ 0,1 algébrico e b irracional algébrico?(ex.: 2 2 {\displaystyle 2^{\sqrt {2}}} ) |
| [8](https://pt.wikipedia.org/wiki/Oitavo_problema_de_Hilbert) | Aberto[8](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | A [Hipótese de Riemann](https://pt.wikipedia.org/wiki/Hip%C3%B3tese_de_Riemann) e a [Conjectura de Goldbach](https://pt.wikipedia.org/wiki/Conjectura_de_Goldbach) |
| [9](https://pt.wikipedia.org/w/index.php?title=Nono_problema_de_Hilbert&action=edit&redlink=1) | Parcialmente  Resolvido[9](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | Achar a lei de reciprocidade mais geral em todo campo de número algébrico |
| [10](https://pt.wikipedia.org/wiki/D%C3%A9cimo_problema_de_Hilbert) | Resolvido[10](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | Encontrar um algoritmo que determine se uma [equação diofantina](https://pt.wikipedia.org/wiki/Equa%C3%A7%C3%A3o_diofantina) tem solução |
| [11](https://pt.wikipedia.org/w/index.php?title=D%C3%A9cimo-primeiro_problema_de_Hilbert&action=edit&redlink=1) | Parcialmente  Resolvido[11](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | Classificar as formas quadráticas a [coeficiente](https://pt.wikipedia.org/wiki/Coeficiente) nos [anéis](https://pt.wikipedia.org/wiki/Anel_(%C3%A1lgebra)) algébricos inteiros |
| [12](https://pt.wikipedia.org/wiki/D%C3%A9cimo_segundo_problema_de_Hilbert) | Aberto[12](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#notas) | Estender o [Teorema de Kronecker-Weber](https://pt.wikipedia.org/wiki/Teorema_de_Kronecker-Weber) para os corpos não [abelianos](https://pt.wikipedia.org/wiki/Extens%C3%A3o_abeliana). |
| [13](https://pt.wikipedia.org/wiki/D%C3%A9cimo-terceiro_problema_de_Hilbert) | Resolvido[13](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | Demonstrar a impossibilidade de resolver equações de sétimo grau através de funções de somente duas variáveis |
| [14](https://pt.wikipedia.org/w/index.php?title=D%C3%A9cimo-quarto_problema_de_Hilbert&action=edit&redlink=1) | Resolvido[14](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | Provar o carácter finito de certos sistemas completos de funções |
| [15](https://pt.wikipedia.org/wiki/D%C3%A9cimo_quinto_problema_de_Hilbert) | Parcialmente  Resolvido[15](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | Desenvolver bases sólidas para o cálculo enumerativo de Schubert |
| [16](https://pt.wikipedia.org/w/index.php?title=D%C3%A9cimo-sexto_problema_de_Hilbert&action=edit&redlink=1) | Aberto[16](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | Desenvolver uma [topologia](https://pt.wikipedia.org/wiki/Topologia_(matem%C3%A1tica)) de curvas e superfícies algébricas |
| [17](https://pt.wikipedia.org/wiki/D%C3%A9cimo-s%C3%A9timo_problema_de_Hilbert) | Resolvido[17](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | Demonstrar que uma [função racional](https://pt.wikipedia.org/wiki/Fun%C3%A7%C3%A3o_racional) positiva pode ser escrita sob a forma de soma de quadrados de funções racionais |
| [18](https://pt.wikipedia.org/w/index.php?title=D%C3%A9cimo-oitavo_problema_de_Hilbert&action=edit&redlink=1) | Resolvido[18](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | Construir um [espaço euclidiano](https://pt.wikipedia.org/wiki/Espa%C3%A7o_euclidiano) com poliedros congruentes. Qual a maneira mais densa de se empacotarem esferas? |
| [19](https://pt.wikipedia.org/w/index.php?title=D%C3%A9cimo-nono_problema_de_Hilbert&action=edit&redlink=1) | Resolvido[19](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | Provar que o cálculo de variações é sempre necessariamente analítico |
| [20](https://pt.wikipedia.org/w/index.php?title=Vig%C3%A9simo_problema_de_Hilbert&action=edit&redlink=1) | Resolvido[20](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | Todos os problemas variacionais com certas condições de contorno têm solução? |
| [21](https://pt.wikipedia.org/wiki/Vig%C3%A9simo-primeiro_problema_de_Hilbert) | Resolvido[21](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | Prova da existência de [equações diferenciais](https://pt.wikipedia.org/wiki/Equa%C3%A7%C3%A3o_diferencial) lineares tendo um determinado grupo monodrômico |
| [22](https://pt.wikipedia.org/w/index.php?title=Vig%C3%A9simo-segundo_problema_de_Hilbert&action=edit&redlink=1) | Resolvido[22](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | Uniformizar as curvas analíticas através de funções automorfas |
| [23](https://pt.wikipedia.org/wiki/Vig%C3%A9simo_terceiro_problema_de_Hilbert) | Aberto[23](https://pt.wikipedia.org/wiki/Problemas_de_Hilbert#Notas) | Desenvolver um método geral de resolução no [cálculo de variações](https://pt.wikipedia.org/wiki/C%C3%A1lculo_de_varia%C3%A7%C3%B5es) |
| [24](https://pt.wikipedia.org/wiki/S%C3%A9timo_problema_de_Hilbert) | Resolvido24 | Hilbert havia listado 24 problemas, mas acabou decidindo não propor um deles. O 24 era sobre um critério para simplicidade e métodos gerais em Teoria de Prova. Deve-se a descoberta deste problema a Rüdiger Thiele, em 2000. |

Já em 1928, Hilbert reformula seu segundo problema (provar a consistência dos “axiomas de Peano”) no Congresso Internacional de Bolonha, Itália, colocando três perguntas:

1. Era a matemática completa? Provar que todas as afirmações verdadeiras matemáticas poderiam ser provadas, isto é, a completude da matemática.
2. Era a matemática consistente? Provar que só as verdadeiras afirmações matemáticas poderiam ser provadas, isto é, a consistência da matemática
3. Era a matemática decidível? Provar a decidibilidade da matemática, isto é, a existência de um processo de decisão para decidir a verdade ou a falsidade de qualquer proposição matemática."

A terceira pergunta ficou conhecida como **Entscheidungsproblem**, o Problema de decisão. A questão retoma a [Gottfried Leibniz](https://pt.wikipedia.org/wiki/Gottfried_Leibniz), que no século XVII, depois de construir uma máquina de calcular mecânica, sonhou em construir uma máquina que pudesse manipular símbolos a fim de determinar os valores de verdade dos enunciados matemáticos. Ele percebeu que o primeiro passo teria que ser uma linguagem formal precisa, e muito do seu trabalho subsequente foi nesse sentido.

É dentro do décimo problema de Hilbert em que a questão de um "Entscheidungsproblem" na verdade aparece. Parecia claro para Hilbert, que com a solução deste problema permitiria ser possível, em princípio, resolver todas as questões matemáticas de uma maneira puramente mecânica. O cerne da questão foi o seguinte questionamento: "O que queremos dizer quando dizemos que uma função é “efetivamente calculável”? A resposta seria algo como: “Quando a função é calculada através de procedimentos mecânicos (processo, método) ”. Apesar de facilmente afirmado hoje em dia, a pergunta seria passaria sem resposta por quase 30 anos antes de ser construída precisamente.

Tese de Church-Turing

A tese que mostraria que o 'Entscheidungsproblem' não pode ser resolvido é conhecida na teoria da computabilidade, como **Tese de Church-Turing**, assim nomeada em referência ao nome dos matemáticos Alonzo Church (1903 - 1995) e Alan Turing (1912 - 1954). Em seu artigo "On Computable Numbers, with an Application to the Entscheidungsproblem", de 1936, Alan Turing tentou capturar a noção de algoritmo (então chamado "computabilidade efetiva"), com a introdução de máquinas de Turing. A máquina de Turing pode ser interpretada como um algoritmo. Geralmente assume-se que um algoritmo deve satisfazer os seguintes requisitos:

* O algoritmo consiste de um conjunto finito de instruções simples e precisas, que são descritas com um número finito de símbolos.
* O algoritmo sempre produz resultado em um número finito de passos.
* O algoritmo pode, a princípio, ser executado por um ser humano com apenas papel e lápis.
* A execução do algoritmo não requer inteligência do ser humano além do necessário para entender e executar as instruções.

Efetivamente toda ação de uma máquina algorítmica (como um computador), se resume a calcular o valor de uma função com determinados argumentos. Um exemplo de tal método é o algoritmo de Euclides para a determinação do máximo divisor comum de dois números naturais. A noção de algoritmo é intuitivamente clara, mas não é definida formalmente, pois não está claro o que quer dizer "instruções simples e precisas", e o que significa "inteligência necessária para executar as instruções".

Informalmente a tese enuncia que nossa noção de algoritmo pode ser formalizada, sob a forma de funções computáveis, e que computadores podem executar esses algoritmos. Além disso, qualquer computador pode, teoricamente, executar qualquer algoritmo, isto é, o poder computacional teórico de cada computador é o mesmo e não é possível construir um artefato de cálculo mais poderoso que um computador e que todos os computadores são "iguais", variando apenas a capacidade de processamento. Este fato é interessante, pois dá uma maneira de se medir a capacidade computacional de uma máquina. Uma máquina que compute mais funções que outra é mais poderosa.

Alguns meses antes do artigo de Turing, Alonzo Church provou um resultado similar em "A Note on the Entscheidungsproblem". Partindo em busca do que seria um procedimento efetivo ou mecânico (o que pode ser feito seguindo-se diretamente um algoritmo ou conjunto de regras), surgiram a sistematização e desenvolvimento das funções recursivas e funções lambda-definíveis para descrever formalmente a computabilidade efetiva.

Usando sua teoria, Church propôs uma formalização da noção de "efetivamente computável", empregando o conceito de lambda-definibilidade, ao invés do computável por uma máquina definida por Turing. Demostrando que lambda-definibilidade é equivalente ao conceito de recursividade de Gödel-Herbrand, o cálculo-lambda (sistema elaborado por Church para ajudar a fundamentar matematicamente o problema) era inconsistente, mas a parte do cálculo-lambda que tratava de funções recursivas estava correta e teve sucesso. Desta forma mostrou que o sistema de procedimentos de Turing era equivalente à lambda-definibilidade. O trabalho de Church e Turing fundamentalmente liga os computadores com as máquinas de Turing. O limite das Maquinas de Turing, de acordo com a tese de Church-Turing, também descreve os limites de todos os computadores.

A partir dos resultados de Kurt Gödel, Alan Turing e Church, pode-se dizer que existem funções para as quais não existe uma sequência de passos que determinem o seu valor, com base nos seus argumentos. Dizendo-se de outra maneira, "não existem algoritmos para a solução de determinadas funções". São as chamadas "funções não computáveis". Isto significa que para tais funções não há nem haverá capacidade computacional suficiente para resolvê-las. Logo, descobrir as fronteiras entre funções computáveis e não computáveis é equivalente a descobrir os limites do computador em geral. A importância disso está na possibilidade de se verificar o alcance e limites de um computador.

A tese de Church-Turing não é um teorema matemático que possa ser provado ou refutado num sentido matemático exato, pois estabelece a identidade de duas noções, em que somente uma (de Church) é definida matematicamente, enquanto a outra (de Turing) é usada pelos matemáticos sem uma definição exata

Embora não haja, para esta tese, uma demonstração, ela é largamente admitida, pois existem fortes argumentos “empíricos” que lhe dão plausibilidade, dentre os quais destacamos:

1. Apesar das inúmeras tentativas, nunca se conseguiu apresentar uma função algorítmica ou parcial algorítmica que não fosse, respectivamente, Turing-computável ou parcialmente Turing-computável.
2. Até hoje, todas as tentativas de caracterizar formalmente as noções vagas e imprecisas de função algorítmica e função parcial algorítmica forneceram exatamente as mesmas classes de funções, a saber, a classe das funções Turing-computáveis e a classe das funções parcialmente Turing-computáveis.

Os argumentos contrários à Tese de Church-Turing, por sua vez, nunca foram fortes o suficiente para refutá-la. Ela é aceita pela maioria dos matemáticos e cientistas da computação e os poucos que demonstram algum grau de descrença quanto ao seu enunciado são tachados excêntricos.

LISP, List Processing

No LISP a [lista](https://pt.wikipedia.org/wiki/Lista) é a [estrutura de dados](https://pt.wikipedia.org/wiki/Estrutura_de_dados) fundamental desta linguagem, tanto os [dados](https://pt.wikipedia.org/wiki/Dados) como os [programa](https://pt.wikipedia.org/wiki/Programa)s são representados como listas, o que permite que a linguagem manipule o [código fonte](https://pt.wikipedia.org/wiki/C%C3%B3digo_fonte) como qualquer outro tipo de dados. Em [1958](https://pt.wikipedia.org/wiki/1958), [John McCarthy](https://pt.wikipedia.org/wiki/John_McCarthy), em um célebre artigo, mostra que é possível usar exclusivamente funções matemáticas como estruturas de dados elementares (o que só foi possível graças ao [Cálculo Lambda](https://pt.wikipedia.org/wiki/C%C3%A1lculo_Lambda), que é um mecanismo formal para manipular funções proposta por [Alonzo Church](https://pt.wikipedia.org/wiki/Alonzo_Church)).

A linguagem Lisp é uma linguagem formal matemática, o que significa que foi projetada primariamente para o processamento de dados simbólicos. Durante os anos de [1970](https://pt.wikipedia.org/wiki/1970) e [1980](https://pt.wikipedia.org/wiki/1980), Lisp se tornou a principal linguagem da comunidade de inteligência artificial, tendo sido pioneiro em aplicações como administração automática de armazenamento, [linguagens interpretadas](https://pt.wikipedia.org/wiki/Interpretador) e [programação funcional](https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_funcional).

A motivação de McCarthfy surgiu da idéia de desenvolver uma linguagem algébrica para processamento de listas para trabalho em IA (inteligência artificial). Esforços para a implementação de seus primeiros dialetos foram empreendidos no [IBM](https://pt.wikipedia.org/wiki/IBM) 704, IBM 7090, DEC PDP-1, DEC PDP-6 e DEC PDP-10. O dialeto principal entre 1960 e 1965 foi o Lisp 1.5. No início dos anos 1970, houve outros dois dialetos predominantes, desenvolvidos através de esforços anteriores: MacLisp e Interlisp.

Scheme

É uma [linguagem de programação](https://pt.wikipedia.org/wiki/Linguagem_de_programa%C3%A7%C3%A3o) [multiparadigma](https://pt.wikipedia.org/wiki/Multiparadigma) que suporta [programação funcional](https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_funcional) e [procedural](https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_procedural) baseado no cálculo lambda.. Foi criada a partir da linguagem Lisp por Guy L. Steele e Gerald Jay Sussman no outono de 1975, quando foi realizada a sua primeira descrição no Laboratório de Inteligência Artificial e Ciência da Computação do MIT (Massachusetts Institute of Technology). Sussman e Steele tiveram alguma dificuldade para entender alguns detalhes da teoria dos atores de Carl Hewitt e suas consequências, então decidiram construir uma implementação de brinquedo de uma linguagem de atores visando experimentá-la.

ML e Standard ML

É uma [linguagem de programação](https://pt.wikipedia.org/wiki/Linguagem_de_programa%C3%A7%C3%A3o) [funcional](https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_funcional) de proposta geral desenvolvida por [Robin Milner](https://pt.wikipedia.org/wiki/Robin_Milner) e outros no final dos [anos 1970](https://pt.wikipedia.org/wiki/Anos_1970) na [Universidade de Edimburgo](https://pt.wikipedia.org/wiki/Universidade_de_Edimburgo), cuja sintaxe foi inspirada pelo [ISWIM](https://pt.wikipedia.org/w/index.php?title=ISWIM&action=edit&redlink=1). Ao contrário de outras linguagens puramente funcionais como [Haskell](https://pt.wikipedia.org/wiki/Haskell_(linguagem_de_programa%C3%A7%C3%A3o)), o ML considerada uma linguagem funcional impura, por permitir a programação imperativa, efeitos colaterais e, por esta razão também é considerada uma linguagem de programação multi-paradigma.

ML foi originalmente concebida como uma metalinguagem para o sistema de prova de teorema de Edimburgo LCF, mas evoluiu para uma linguagem de propósito geral de sucesso. Esta linguagem foi padronizada em 1990 e revista em 1997 como Standard ML 97.

As características do ML são incluir chamada-a-valor avaliação e estratégia, em primeira classe funções de gerenciamento automático de memória através de coleta de lixo, polimorfismo paramétrico, tipagem estática, tipo inferência, tipos de dados algébricos, correspondência padrão, exceção e manuseio.

Os pontos fortes da ML são aplicados principalmente em língua e manipulação (compiladores, analisadores, provadores de teoremas), mas é uma linguagem de aplicação geral também utilizado em bioinformática, sistemas financeiros, e aplicativos, incluindo um banco de dados genealógicos, um clienteP2P / programa servidor, etc..

Hoje existem vários idiomas no ML família; os dois principais dialetos são Standard ML e Caml, mas existem outros que influenciaram muitas outras línguas, como Haskell, Cyclone, e Nemerle.

Standard ML foi criada por pesquisadores da LFCS (Laboratory for Foundations of Computer Science) na década de 1980. Em 1987, Robin Milner e LFCS ganharam o Prêmio BCS Award for Technical Excellence por trabalhar no Standard ML.

OCaml - Objective Caml

Caml, acrónimo de "Categorical Abstract MetaLanguage", é um dialeto da família de linguagens de programação do ML, desenvolvida pelo INRIA e pela École Normale Supérieure, em Paris. Como todos os descendentes do ML, Caml é uma linguagem de tipagem estática, de avaliação estrita (strict evaluation) e usa gerenciamento automático de memória.

A primeira implementação do Caml foi escrita em Lisp em 1987 foi apelidada de "Heavy CAML" (CAML pesado) por causa dos requisitos de CPU e memória excessivos se comparados com seu sucessor, o Caml light (Caml leve), no início dos anos 90, que foi implementado em linguagem C por Xavier Leroy e Damien Doligez. Além de ser completamente reescrito, o CAML special Light acrescentou um poderoso sistema de módulos na base da linguagem.

O Caml light é obsoleto e não é mais mantida ativamente. Atualmente a principal implementação do Caml é o Objective Caml, que acrescenta várias novas funcionalidades à linguagem, incluindo uma camada de orientação a objetos e coletor de lixo, desenvolvida pelo [INRIA](https://pt.wikipedia.org/wiki/Institut_National_de_Recherche_en_Informatique_et_en_Automatique) em [1996](https://pt.wikipedia.org/wiki/1996).

Haskell

É uma [linguagem de programação](https://pt.wikipedia.org/wiki/Linguagem_de_programa%C3%A7%C3%A3o) puramente [funcional](https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_funcional), de propósito geral, nomeada em homenagem ao lógico [Haskell Curry](https://pt.wikipedia.org/wiki/Haskell_Curry). Como uma linguagem funcional, a estrutura de controle primária é a função; a linguagem é baseada nas observações de Haskell Curry e seus descendentes intelectuais.[[4]](https://pt.wikipedia.org/wiki/Haskell_(linguagem_de_programa%C3%A7%C3%A3o)#cite_note-4)[[5]](https://pt.wikipedia.org/wiki/Haskell_(linguagem_de_programa%C3%A7%C3%A3o)#cite_note-5) Seu último padrão semi-oficial é o Haskell 98, destinado a especificar uma versão mínima e portável da linguagem para o ensino e como base para futuras extensões.

A primeira versão de Haskell foi definida em [1 de abril](https://pt.wikipedia.org/wiki/1_de_abril) de [1990](https://pt.wikipedia.org/wiki/1990).[[9]](https://pt.wikipedia.org/wiki/Haskell_(linguagem_de_programa%C3%A7%C3%A3o)#cite_note-9) Seguiu-se a versão 1.1 em agosto do ano seguinte, a versão 1.2 em março de 1992, a versão 1.3 em maio de 1996 e a versão 1.4 em abril de 1997. Esforços posteriores culminaram no Haskell 98, publicado em janeiro de 1999 e que especifica uma versão mínima, estável e portável da linguagem e a [biblioteca](https://pt.wikipedia.org/wiki/Biblioteca_(computa%C3%A7%C3%A3o)) para ensino. Esse padrão sofreu uma revisão em janeiro de 2003.

O conceito de [avaliação preguiçosa](https://pt.wikipedia.org/wiki/Avalia%C3%A7%C3%A3o_pregui%C3%A7osa) já estava difundido no meio acadêmico desde o final da [década de 1970](https://pt.wikipedia.org/wiki/D%C3%A9cada_de_1970). Esforços nessa área incluíam técnicas de [redução de grafo](https://pt.wikipedia.org/w/index.php?title=Redu%C3%A7%C3%A3o_de_grafo&action=edit&redlink=1) e a possibilidade de uma mudança radical na [arquitetura de von Neumann](https://pt.wikipedia.org/wiki/Arquitetura_de_von_Neumann). Após o lançamento de [Miranda](https://pt.wikipedia.org/wiki/Miranda_(linguagem_de_programa%C3%A7%C3%A3o)) em [1985](https://pt.wikipedia.org/wiki/1985), diversas outras linguagens funcionais de semântica não rígida proliferaram, como Lazy ML, Orwell, Alfl, Id, Clean, Ponder e Daisy (um dialeto de [Lisp](https://pt.wikipedia.org/wiki/Lisp)). Mesmo após dois anos, Miranda ainda era a mais usada, mas não estava em domínio público. Haskell foi criada da necessidade de unir as outras linguagens do mesmo paradigma em uma só.

Em setembro 1987 foi realizada uma conferência Functional Programming Languages and Computer Architecture (FPCA '87), no Oregon. O consenso foi a criação de um comitê com o objetivo de construir um [padrão aberto](https://pt.wikipedia.org/wiki/Padr%C3%A3o_aberto) para tais linguagens. Isso consolidaria as linguagens existentes, servindo como base para pesquisas futuras no desenvolvimento de linguagens. A primeira reunião do comitê foi realizada em janeiro de 1988, e dentre os principais objetivos deste comitê, ao projetar essa nova linguagem, estavam suas especificações:

* Ser viável para o ensino, pesquisa e aplicações, incluindo sistema de larga escala;
* Ser completamente descritiva via publicação no tocante à sua sintaxe e sua semântica;
* Não ser proprietária, tal que qualquer um pudesse implementá-la e distribuí-la;
* Basear-se em ideias que envolvessem o senso comum;
* Reduzir a diversidade desnecessária de outras linguagens funcionais.

F#

É uma [linguagem de programação](https://pt.wikipedia.org/wiki/Linguagem_de_programa%C3%A7%C3%A3o) [multiparadigma](https://pt.wikipedia.org/wiki/Paradigma_de_programa%C3%A7%C3%A3o) para a plataforma [.NET](https://pt.wikipedia.org/wiki/Microsoft_.NET) surgido em 2005, que engloba [programação funcional](https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_funcional), [imperativa](https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_imperativa) e [orientada a objetos](https://pt.wikipedia.org/wiki/Orienta%C3%A7%C3%A3o_a_objetos). Pertence à família das [linguagens ML](https://pt.wikipedia.org/wiki/ML_(linguagem_de_programa%C3%A7%C3%A3o)), e é em grande parte compatível com a linguagem [OCaml](https://pt.wikipedia.org/wiki/Objective_Caml). Foi inicialmente desenvolvida por Don Syme na [Microsoft Research](https://pt.wikipedia.org/wiki/Microsoft_Research), e então migrou para a Microsoft Developer Division. É distribuida como parte do [Visual Studio](https://pt.wikipedia.org/wiki/Microsoft_Visual_Studio).

As propriedades desta linguagem e o seu ambiente constituem uma plataforma ideal para a informática científica porque utiliza a inferência de tipo. A performance de F# iguala-se com a linguagem [OCaml](https://pt.wikipedia.org/wiki/OCaml), [C++](https://pt.wikipedia.org/wiki/C%2B%2B) e ML. É uma linguagem criada de propósito para a plataforma .NET e portanto é orientada a objetos.

Programação Funcional em Python

## O Python suporta paradigmas de Programação Funcional, ou PF, sem o suporte de quaisquer recursos ou bibliotecas especiais. A programação funcional, é um paradigma de codificação no qual os blocos de construção são valores imutáveis ​​e "funções puras" que não compartilham nenhum estado com outras funções. Toda vez que uma função pura possui uma determinada entrada, ela retornará a mesma saída - sem alterar dados ou causar efeitos colaterais. Nesse sentido, funções puras são frequentemente comparadas a operações matemáticas. Por exemplo, 3 mais 4 sempre serão iguais a 7, independentemente de quais outras operações matemáticas estejam sendo feitas ou quantas vezes você adicionou coisas antes.

Embora a programação funcional exista desde a década de 1950 e seja implementada por uma longa linhagem de linguagens, ela não descreve completamente uma linguagem de programação. Clojure, Common Lisp, Haskell e OCaml apesar de serem consideradas linguagens primariamente funcionais, possuem influências de outros conceitos de linguagem de programação, como sistema de tipos e de avaliação estrita ou lenta. A maioria deles também suporta efeitos colaterais, como escrever e ler arquivos de uma maneira ou de outra, e por isso são consideradas impuras.

O florescer da programação funcional nasce no Lisp (acrônomo para List Processing) para tentar resolver alguns problemas de inteligência artificial que eram provenientes da linguística, que tinha foco em processamento de linguagem natural que por sua vez eram focados em processamento de listas em geral. Isso justifica uma grande parte do conteúdo que vamos ver aqui e seus tipos de dados variam somente entre listas e átomos. E assim foi mantido o foco de processamento de listas em todas as linguagens funcionais e suas funções e abstrações para resolver problemas relativos a listas e estruturas iteráveis.

A programação funcional pode ter reputação de ser obscura e de favorecer a elegância ou a concisão sobre a praticidade. As grandes empresas raramente dependem de linguagens funcionais em primeira escala, ou pelo menos em um nível menor do que outras linguagens como C++, Java ou Python. O FP, no entanto, é realmente apenas uma estrutura para pensar em fluxos lógicos, com suas vantagens e desvantagens, e é passível de composição com outros paradigmas.

A programação funcional é um tipo declarativo de estilo de programação. Seu foco principal é "o que resolver", em contraste com um estilo imperativo, onde o foco principal é "como resolver". Ele usa expressões em vez de instruções. Uma expressão é avaliada para produzir um valor, enquanto uma instrução é executada para atribuir variáveis.

Vantagens da programação funcional incluem:

* **Código mais limpo:** "variáveis" não são modificadas depois de definidas, portanto, não precisamos seguir a mudança de estado para compreender o que funciona como uma função, um método, uma classe, um projeto inteiro.
* **Transparência referencial:** as expressões podem ser substituídas por seus valores. Se chamarmos uma função com os mesmos parâmetros, sabemos com certeza que a saída será a mesma (não há estado em nenhum lugar que a altere). Vantagens possibilitadas pela transparência referencial

|  |  |
| --- | --- |
| **Vantagem** | **Descrição** |
| Memoização | Resultados de cache para chamadas de função anteriores. Embora relacionadas com [caching](https://pt.qwe.wiki/wiki/Cache_(computing)), refere-se a um caso específico dessa otimização |
| Idempotência | Os mesmos resultados, independentemente de quantas vezes você chama uma função. |
| Modularização | Como não temos um estado que permeie todo o código, construímos nosso projeto com pequenas caixas pretas que amarramos para promover a programação de baixo para cima. |
| Facilidade de depuração | As funções são isoladas, elas dependem apenas de suas entradas e saídas, portanto, são muito fáceis de depurar. |
| Paralelização | As chamadas de funções são independentes. Podemos paralelizar em diferentes processos / CPUs / computadores /…  resultado = func1 (a, b) + func2 (a, c).  Podemos executar func1 e func2 em paralelo porque não serão modificados. |
| Concorrência | Sem dados compartilhados, a concorrência fica muito mais simples:   * Não há necessidade de usar semáforo, monitores e bloqueios para sincronizar a ação dos dois threads. * Não há race condition que é uma situação indesejável que ocorre quando um dispositivo ou sistema tenta executar duas ou mais operações ao mesmo tempo, mas devido à natureza do dispositivo ou sistema, as operações devem ser executadas na seqüência adequada para serem executadas corretamente. * Não há Deadlock que é um erro de concorrência comum, que ocorre quando um um conjunto de threads está bloqueado, devido a cada tentativa de adquirir um bloqueio retido por outro. |

Abaixo algumas operações em objetos de primeira ordem, comparando a abordagem imperativa (com mudança de estado) e a funcional.

*# Not Functional - STRING*name = **"Geison"**name = **"{0} Flores"**.format(name)  
print(name)  
  
*# Functional*firstname = **"Geison"**lastname = **"Flores"**name = **"{0} {1}"**.format(firstname, lastname)  
print(name)  
  
*# Not Functional - LIST*years = [2001, 2002]  
years.append(2003)  
years += [2004, 2005]  
print(years) *# [2001, 2002, 2003, 2004, 2005]  
  
# Functional*years = [2001, 2002]  
all\_years = years + [2003] + [2004, 2005]  
print(all\_years)  
  
*# Not Functional - DICT*ages = {**"John"**: 30}  
ages[**"Mary"**] = 28  
print(ages) *# {‘John’: 30, ‘Mary’: 28}  
  
# Functional*ages = {**"John"**: 30}  
all\_ages = dict(ages, \*\*{**"Mary"**: 28})  
print(all\_ages) *# {‘John’: 30, ‘Mary’: 28}*

Avaliação estrita e não estrita

As linguagens funcionais podem ser categorizadas em a como os argumentos da função são processados ​​quando uma expressão está sendo avaliada que dividem-se em dois tipos:

1. **Avaliação estrita (ansiosa/rigorosa):** usado pela maioria das tradicionais [linguagens de programação](https://pt.qwe.wiki/wiki/Programming_language), onde uma [expressão](https://pt.qwe.wiki/wiki/Expression_(mathematics)) é avaliada logo que ele está ligado a uma [variável](https://pt.qwe.wiki/wiki/Variable_(programming)) .
2. **Avaliação não estrita (preguiçosa/lenta):** as expressões são avaliados apenas quando uma expressão dependente é avaliada dependendo de uma definida [estratégia de avaliação](https://pt.qwe.wiki/wiki/Evaluation_strategy#Non-strict_evaluation) .

Linguagens imperativas de programação, onde a ordem de execução é definida implicitamente pela estrutura do código-fonte, quase sempre usam avaliação ansiosa, a fim de evitar um comportamento inesperado que pode ocorrer em certos contextos de execução fora de ordem (por exemplo, quando se utiliza software multithreaded, execução simultânea de código, etc.). Este comportamento inesperado pode resultar em problemas de race condition, violações de atomicidade e outros erros potencialmente indesejados e difícil de controlar os efeitos.

A diferença técnica está na semântica denotacional de expressões que contêm cálculos com falha ou divergentes. Sob avaliação estrita, a avaliação de qualquer termo que contém um subtermo com erro, falha. Ou seja, a avaliação rigorosa sempre avalia totalmente os argumentos da função antes de chamar a função. Por exemplo, a expressão len([2 + 1, 3 \* 2, 1/0, 5-4]) falha sob avaliação rigorosa devido à divisão por zero no terceiro elemento da lista.

Sob avaliação preguiçosa, a função length retorna o valor 4 (ou seja, o número de itens na lista), pois a avaliação não tenta avaliar os termos que compõem a lista. Em resumo, a avaliação rigorosa sempre avalia totalmente os argumentos da função antes de chamar a função. A avaliação preguiçosa não avalia argumentos de função, a menos que seus valores sejam necessários para avaliar a própria chamada de função.

Avaliação preguiçosa é usado particularmente na [programação funcional](https://pt.qwe.wiki/wiki/Functional_programming) por terem a vantagem de ser capaz de criar listas infinitas calculáveis sem loops infinitos ou limtasçoes de tamanho da lista que interfiram no cálculo. Por exemplo, pode-se criar funções que contêm a palavra reservada yield que cria uma lista infinita (muitas vezes chamado de [fluxo](https://pt.qwe.wiki/wiki/Stream_(computing))) de [números de Fibonacci](https://pt.qwe.wiki/wiki/Fibonacci_number). O cálculo do n -ésimo número de Fibonacci seria apenas a extração desse elemento da lista infinita, forçando a avaliação de apenas os primeiros n membros da lista.

Avaliação preguiçosa é muitas vezes combinada com [memoização](https://pt.qwe.wiki/wiki/Memoization) (Embora relacionadas com [caching](https://pt.qwe.wiki/wiki/Cache_(computing)), refere-se a um caso específico dessa otimização). Após o valor de uma função ser calculado para esse parâmetro ou conjunto de parâmetros, o resultado é armazenado em uma tabela de pesquisa que é indexado pelos valores desses parâmetros; da próxima vez que a função é chamada, a tabela é consultada para determinar se o resultado para essa combinação de valores de parâmetros já está disponível. Se assim for, o resultado armazenado é simplesmente retornado. Se não, a função é avaliada e outra entrada é adicionada à tabela de pesquisa para reutilização.

Em [Python](https://pt.qwe.wiki/wiki/Python_(programming_language)) 2.x, qunado a função range() calculava uma lista de inteiros, a lista completa era armazenada na memória quando a primeira instrução de atribuição é avaliada, de modo que era exemplo de avaliação ansiosa ou imediata:

r = range(10)  
print(r) *#[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]*print(r[3]) *# 3*

Já no Python 3.x a função range() passou a retornar um objeto de intervalo especial que calcula elementos da lista na procura. Elementos do objecto intervalo são gerados apenas quando eles forem necessários (por exemplo, quando print(r[3])é avaliada, de modo que a função se tornou um exemplo de avaliação preguiçosa:

r = range(10)  
print(r) *# range(0, 10) Lazy evaluation*print(r[3]) *# 3*

Esta mudança para avaliação preguiçosa economiza tempo de execução para grandes faixas que nunca pode ser totalmente referenciados e uso de memória para grandes intervalos em que apenas um ou alguns elementos são necessários a qualquer momento.

Em Python 2.x também era possível utilizar uma função chamada xrange() que devolvia um objecto que gera os números na gama de procura. A vantagem xrange era que objeto gerado sempre teria a mesma quantidade de memória, de modo que era outro exemplo de avaliação ansiosa.

r = xrange(10)  
print(r) *# xrange(10)*lst = [x **for** x **in** r]  
print(lst) *# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]*

A partir da versão 2.2 para a frente, Python manifesta avaliação preguiçosa através da implementação de iterators ao contrário de seqüências tupla ou lista (seqüências ansiosas):

numbers = range(10)  
iterator = iter(numbers)

print(numbers) *# range(0, 10)*

print(list(numbers)) *# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]*print(iterator) *# <range\_iterator object at 0x0000018AE309E770>*print(next(iterator)) *# 0*print(next(iterator)) *# 1*new\_iter = iter(numbers)  
print(new\_iter) *# <range\_iterator object at 0x0000018AE303C4D0>*print(next(new\_iter)) *# 0*print(next(new\_iter)) *# 1*print(next(new\_iter)) *# 2*

O exemplo acima mostra que as listas são avaliadas quando chamada, mas em caso de iteração, o primeiro elemento de '0' é impresso quando a necessidade surge. Mas tome cuidado para não utilizar iter() para todas as chamadas de next(), pois ele acaba criando um novo iterador no índice 0.

## POO vs PF

A maneira mais fácil de introduzir programação funcional é compará-la com algo que já sabemos: programação orientada a objetos. Suponha que desejássemos criar uma classe de contador de linha que capte um arquivo, leia cada linha e conte a quantidade total de linhas no arquivo. Usando uma classe, pode ser algo como o seguinte:

**class** LineCounter:  
 **def** \_\_init\_\_(self, filename):  
 self.file = open(filename, **'r'**, encoding=**"utf8"**)  
 self.lines = []  
 **def** read(self):  
 self.lines = [line **for** line **in** self.file]  
 **def** count(self):  
 **return** len(self.lines)  
  
arquivo = LineCounter(**'nome\_errado.txt'**)  
print(arquivo.lines) *# []*print(arquivo.count()) *# 0  
  
# The lc object must read the file to  
# set the lines property.*arquivo.read()  
*# The `lc.lines` property has been changed.  
# This is called changing the state of the lc object.*print(arquivo.lines) *# 'Toque Frágil (Walter Franco)\n', '===*print(arquivo.count()) *# 7*

A implementação fornece uma visão do design orientado a objetos. Dentro da classe, existem os conceitos familiares de métodos e propriedades. As propriedades configuram e recuperam o estado do objeto e os métodos manipulam esse estado. Para que esses dois conceitos funcionem, o estado do objeto deve mudar com o tempo. Essa mudança de estado é evidente na propriedade lines depois de chamar o método read(). Como exemplo de programação funcional, onde haveria o isolamento dos parâmetros de entrada e saída, usaríamos:

**def** LineCounter(filename):  
 **with** open(filename, **'r'**) **as** f:  
 **return** [line **for** line **in** f]  
  
**def** count(lines):  
 **return** len(lines)  
  
arquivo = LineCounter(**'nome\_errado.txt'**)  
print(count(arquivo)) *# 7*print(arquivo) *# ['Toque Frágil (Walter*print(count(arquivo)) *# 7*print(arquivo) *# ['Toque Frágil (Walter*

Adaptando a programação POO para corrigir o erro de atribuição do primeiro exemplo, poderíamos ter o seguinte:

**class** LineCounter:  
 **def** \_\_init\_\_(self, filename):  
 **with** open(filename, **'r'**, encoding=**"utf8"**) **as** file:  
 self.lines = [line **for** line **in** file]  
 **def** count(self):  
 **return** len(self.lines)  
  
arquivo = LineCounter(**'nome\_errado.txt'**)  
  
print(**"Linhas: "** + str(arquivo.count())) *# Linhas: 7*print(arquivo.lines) *# ['Toque Frágil (Walter mmm*print(**"Linhas: "** + str(arquivo.count())) *# Linhas: 7*print(arquivo.lines) *# ['Toque Frágil (Walter*

Outro exemplo comparativo entre POO e Fncional:

*# Imperative*n, num\_elements, s = 1, 0, 0  
**while** num\_elements < 10:  
 **if** n\*\*2 % 5 == 0:  
 s += n  
 num\_elements += 1  
 n += 1  
  
print(s) *# 275  
  
# Functional*soma = sum(list(filter(**lambda** x: x\*\*2 % 5 == 0, range(1, 100)))[:10])  
print(s) *# 275*

Linguagens orientadas a objetos são boas quando você tem um conjunto fixo de operações e, à medida que seu código evolui, você adiciona principalmente coisas novas. Isso pode ser conseguido adicionando novas classes que implementam métodos existentes e as classes existentes são deixadas em paz.

Linguagens funcionais são boas quando você tem um conjunto fixo de coisas e, à medida que seu código evolui, você adiciona principalmente novas operações às coisas existentes. Isso pode ser conseguido adicionando novas funções que computam com os tipos de dados existentes, e as funções existentes são deixadas em paz.

|  |  |
| --- | --- |
| **Programação Funcional** | **Programação orientada por objetos** |
| Enfatiza o uso de funções em que cada função executa uma tarefa específica. | É baseado no conceito orientado a objetos. As classes são usadas onde a instância de objetos é criada |
| Os elementos fundamentais utilizados são variáveis e funções. Os dados nas funções são imutáveis . | Os elementos fundamentais usados são objetos e métodos e os dados usados aqui são dados mutáveis. |
| Segue modelo de programação declarativa. | Segue modelo de programação imperativa. |
| Ele usa recursão para iteração. | Ele usa loops para iteração. |
| É programação paralela suportada. | Não suporta programação paralela. |
| As instruções não precisam seguir uma ordem específica durante a execução. | As instruções precisam seguir uma ordem, isto é, uma abordagem ascendente durante a execução. |

Qualquer linguagem de programação funcional deve seguir esses conceitos.

1. **As variáveis são imutáveis**: na programação funcional, não podemos modificar uma variável depois que ela foi inicializada. Podemos criar novas variáveis, mas não podemos modificar as variáveis existentes.
2. **Funções puras:** Essas funções têm duas propriedades principais. Primeiro, eles sempre produzem a mesma saída para os mesmos argumentos, independentemente de qualquer outra coisa. Em segundo lugar, eles não têm efeitos colaterais, ou seja, modificam qualquer argumento ou variável global ou produzem algo.
3. **Recursão:** Não há loop "for" ou "while" nas linguagens funcionais. A iteração em linguagens funcionais é implementada através da recursão. A função chama a si mesma, com novas entradas, até que os parâmetros atendam a uma condição de terminação.
4. **As funções são de primeira classe e podem ser de ordem superior:** as funções de primeira classe são tratadas como variáveis de primeira classe. As variáveis de primeira classe podem ser passadas para funções como um parâmetro, podem ser retornadas de funções ou armazenadas em estruturas de dados.

Imutabilidade

A imutabilidade faz sentido dentro da programação funcional pelo seu viés matemático. Imutabilidade significa que uma vez que uma variável que recebeu um valor, vai possuir esse valor para sempre, ou quando criamos um objeto ele não pode ser modificado, independentemente de onde esteja ou como está sendo usado.

A imutabilidade pode ser usada também para depuração, pois gera um erro em que a variável está sendo alterada e não em que o valor é alterado. O Python também suporta alguns tipos de dados imutáveis, como string, tupla, numérico etc.

*# Python program to demonstrate   
# immutable data types   
  
# String data types*immutable = **"GeeksforGeeks"***# changing the values will raise an error  
# TypeError: 'str' object does not support item assignment*immutable[1] = **'K'**

## Funções puras

É uma função onde o resultado é apenas determinado pelos seus parâmetros de entrada, sem depender de fatores externos (APIs, sistema de arquivos, qualquer dependência que possa alterar o resultado de uma função). Dessa forma, uma função pura retornará sempre o mesmo resultado para os mesmos parâmetros de entrada.

## Funções puras têm duas propriedades.

## Sempre produz a mesma saída para os mesmos argumentos. Por exemplo, 3 + 7 sempre será 10, não importa o quê.

## Não altera ou modifica a variável de entrada, também é conhecido como imutabilidade.

Programas feitos usando programação funcional são fáceis de depurar, porque funções puras não têm efeitos colaterais ou E/S ocultas. As funções puras também facilitam a gravação de aplicativos paralelos/simultâneos. Quando o código é escrito nesse estilo, um compilador inteligente pode fazer muitas coisas - pode paralelizar as instruções, aguardar para avaliar os resultados quando necessário e memorizar os resultados, pois os resultados nunca mudam desde que a entrada não seja alterada.

Códigos previsíveis e sem efeitos colaterais nos ajudam a desenvolver de forma que múltiplos pedaços do nosso sistema rode em múltiplos processadores. Se não houver dependência de dados entre duas expressões puras, sua ordem pode ser revertida ou elas podem ser executadas em [paralelo](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&pto=nl&rurl=translate.google.com.br&sl=auto&sp=nmt4&tl=pt&u=https://en.m.wikipedia.org/wiki/Parallelization&usg=ALkJrhhham2UZMdlOvfDtv3gzN_NGm8oFA) e não podem interferir uma com a outra (em outros termos, a avaliação de qualquer expressão pura é [segura para threads](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&pto=nl&rurl=translate.google.com.br&sl=auto&sp=nmt4&tl=pt&u=https://en.m.wikipedia.org/wiki/Thread-safe&usg=ALkJrhjVdRBBTDRc8A0Z04LdhFELKOyWPg)).

**def** pure\_func(List):  
 New\_List = []  
 **for** i **in** List:  
 New\_List.append(i \*\* 2)  
 **return** New\_List  
  
Original\_List = [1, 2, 3, 4]  
Modified\_List = pure\_func(Original\_List)  
  
*# Original List: [1, 2, 3, 4]*print(**"Original List:"**, Original\_List)  
*# Modified List: [1, 4, 9, 16]*print(**"Modified List:"**, Modified\_List)

Outro exemplo, onde se compara uma função impura e uma pura:

A = 5  
  
**def** impure\_sum(b):  
 *# Adds two numbers, but uses the  
 # global `A` variable.* **return** b + A  
  
**def** pure\_sum(a, b):  
 *# Adds two numbers, using  
 # ONLY the local function inputs.* **return** a + b  
  
print(impure\_sum(5)) *# 10*print(pure\_sum(4, 6)) *# 10*

Essas funções podem ser aplicadas também em sequências de dados, geralmente listas. Imagine uma lista de preços em que você precisa adicionar 10% em todos os produtos da lista.

*# Abordagem tradicional Imperativo*prices = [50, 60, 70]  
newPrices = []  
**for** price **in** prices:  
 newPrice = price \* 1.1  
 newPrices.append(newPrice)  
print (newPrices) *# [55.00000000000001, 66.0, 77.0]  
  
# Função pura Funcional*prices = [50, 60, 70]  
list(map(**lambda** price: price \* 1.1 , prices))  
print (prices) *# [50, 60, 70]*

Uma função pura deve funcionar como uma caixa preta, todas as vezes em que o mesmo input for dado nela, ela terá que retornar o mesmo valor. O benefício de usar funções puras sobre funções impuras (não puras) é a redução dos efeitos colaterais. Os efeitos colaterais ocorrem quando há alterações realizadas dentro da operação de uma função que estão fora de seu escopo. Por exemplo, eles ocorrem quando alteramos o estado de um objeto:

valor = 5  
  
**def** mais\_cinco(x):  
 **return** x + valor  
  
**assert** mais\_cinco(5) == 10 *# True*valor = 7  
**assert** mais\_cinco(5) == 10 *# AssertionError*

Ou, quando realizamos qualquer operação de E/S ou chamamos print():

**def** read\_and\_print(filename):  
 **with** open(filename, **'r'**, encoding=**"utf8"**) **as** f:  
 *# Side effect of opening a  
 # file outside of function.* data = [line **for** line **in** f]  
 **for** line **in** data:  
 *# Side effect of Call out to*

*# the operating system "println" method.* print(line)  
  
read\_and\_print(**'nome\_errado.txt'**)

Os programadores reduzem os efeitos colaterais em seu código para facilitar o acompanhamento, teste e depuração. Quanto mais efeitos colaterais uma base de código tiver, mais difícil será percorrer um programa e entender sua sequência de execução.

Embora seja conveniente tentar eliminar todos os efeitos colaterais, eles costumam ser usados para facilitar a programação. Como fazemos interface com o mundo real, se proibíssemos todos os efeitos colaterais, você não conseguiria ler um arquivo, ligar para imprimir ou atribuir uma variável a uma função. Os advogados de programação funcional entendem essa compensação e tentam eliminar os efeitos colaterais sempre que possível, sem sacrificar o tempo de implementação do desenvolvimento.

Recursão

Durante a programação funcional, não há conceito de loop for ou loop while, em vez disso, recursão é usada. A recursão é um processo no qual uma função se chama direta ou indiretamente. No programa recursivo, a solução para o caso base é fornecida e a solução para o problema maior é expressa em termos de problemas menores. Pode surgir uma pergunta: qual é o caso base? O caso base pode ser considerado como uma condição que informa ao compilador ou intérprete para sair da função.

Exemplo: vamos considerar um programa que encontrará a soma de todos os elementos de uma lista sem usar nenhum loop for.

**def** Sum(L, i, n, count):  
 *# Base case* **if** n <= i:  
 **return** count  
 count += L[i]  
  
 *# Going into the recursion* count = Sum(L, i + 1, n, count)  
 **return** count  
  
L = [1, 2, 3, 4, 5]  
count = 0  
n = len(L)  
print(Sum(L, 0, n, count)) *# 15*

O exemplo mais popular de recursão é o cálculo do fatorial. Matematicamente, o fatorial é definido como: n! = n \* (n-1)! Usamos o próprio fatorial para definir o fatorial. Portanto, este é um caso adequado para escrever uma função recursiva. Vamos expandir a definição acima para o cálculo do valor fatorial de 5.

5! = 5 X 4!

5 X4 X 3!

5 X4 X 3 X 2!

5 X4 X 3 X 2 X 1!

5 X4 X 3 X 2 X 1

= 120

Embora possamos realizar esse cálculo usando um loop, sua função recursiva envolve chamá-lo sucessivamente, decrementando o número até atingir 1. A seguir, é uma função recursiva para calcular o fatorial.

**def** factorial(n):  
 **if** n == 1:  
 print(n)  
 **return** 1  
 **else**:  
*# 14 \* 13 \* 12 \* 11 \* 10 \* 9 \* 8 \* 7 \* 6 \* 5 \* 4 \* 3 \* 2 \* 1* print(n, **'\*'**, end=**' '**)  
 **return** n \* factorial(n - 1)  
  
*# 87178291200*print(factorial(14))

Quando a função fatorial é chamada, são feitas chamadas sucessivas para a mesma função, reduzindo o valor. As funções começam a retornar à chamada anterior depois que o argumento atinge 1. O valor de retorno da primeira chamada é cumulativo produto dos valores de retorno de todas as chamadas.

## Funções de Primeira Classe

Funções como objeto de primeira classe, são funções que se comportam como qualquer tipo nativo de uma determinada linguagem. Por exemplo: lista = [1, 'str', [1,2], (1,2), {1,2}, {1: 'um'}]

## Todos esses exemplos são tipos de objetos de primeira classe em Python, e são tratados uniformemente por toda parte. Eles podem ser armazenados em estruturas de dados, passados como argumentos ou usados em estruturas de controle. Diz-se que uma linguagem de programação suporta funções de primeira classe se tratar funções como objetos de primeira classe. Você deve ser capaz de fazer com uma função o que você pode fazer com valores dos outros tipos internos e ser capaz de fazer isso com um grau de esforço comparável.

## Propriedades das funções de primeira classe:

## Uma função é uma instância do tipo de objeto.

## Você pode armazenar a função em uma variável.

## Você pode passar a função como parâmetro para outra função.

## Você pode retornar uma função de uma chamada de função.

## Você pode armazená-los em estruturas de dados como tabelas de hash, listas,…

**def** soma(x, y): **return** x + y  
  
*#<class 'function'>*print(type(soma))  
  
s = soma  
print(s(1, 2)) *# 3*print(soma(1, 2)) *# 3*

## Abaixo, exemplo comparativo utilizando duas funções:

**def** shout(text):  
 **return** text.upper()  
  
**def** whisper(text):  
 **return** text.lower()  
  
**def** greet(func):  
 *# storing the function in a variable* greeting = func(**"Hi, I am created by a function passed as an argument."**)  
 print(greeting)  
  
*#HI, I AM CREATED BY A FUNCTION PASSED AS AN ARGUMENT.*greet(shout)  
*#hi, i am created by a function passed as an argument.*greet(whisper)

Funções internas de ordem superior

Para facilitar o processamento de objetos iteráveis, como listas e iterador, o Python implementou algumas funções de ordem superior comumente usadas. Essas funções retornam um iterador com economia de espaço.

Funções de ordem superior estão intimamente relacionadas às [funções de primeira classe](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&pto=nl&rurl=translate.google.com.br&sl=auto&sp=nmt4&tl=pt&u=https://en.m.wikipedia.org/wiki/First-class_function&usg=ALkJrhhdSoah_weuX1cvujkdDxP76Dd6gw), pois funções de ordem superior e funções de primeira classe permitem funções como argumentos e resultados de outras funções. A distinção entre os dois é sutil: "ordem superior" descreve um conceito matemático de funções que operam em outras funções, enquanto "primeira classe" é um termo de ciência da computação que descreve entidades da linguagem de programação que não têm restrição de uso. Portanto, funções de primeira classe podem aparecer em qualquer lugar do programa que outras entidades de primeira classe (como números) incluindo argumentos para outras funções e como seus valores de retorno. No cálculo, um exemplo de função de ordem superior é o [operador diferencial](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&pto=nl&rurl=translate.google.com.br&sl=auto&sp=nmt4&tl=pt&u=https://en.m.wikipedia.org/wiki/Differential_operator&usg=ALkJrhjheEaApfnVM6Jqoi9qB7jIRKLjxA)

, que retorna a [derivada](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&pto=nl&rurl=translate.google.com.br&sl=auto&sp=nmt4&tl=pt&u=https://en.m.wikipedia.org/wiki/Derivative&usg=ALkJrhhJPImcRbWzgX7C16Msk1g_cBOhQg) de uma função

* Padrões de computação comuns, tais como aplicar uma função duas vezes, podem ser naturalmente expressos como funções de ordem superior de propósito geral.
* Linguagens de domínio específico podem ser definidas em Haskell usando funções de ordem superior. Por exemplo: processamento de listas, parsing, interação.
* Propriedades algébricas de funções de ordem superior podem ser usadas para raciocinar sobre programas.

**def** caller(f):  
 **return** f()  
  
**def** say\_hello():  
 **return "Hello {0}"**.format(**"Baby"**)  
  
print(caller(say\_hello)) *# Hello Baby*

Exemplo que ilustra o quão mais sucinto fica o código utilizando funções de ordem superior:

*# Programação Imperativa*lista = (0.1, 0.3, 2.5, 0.8, 1.1, 0.1)  
menores\_que\_1 = []  
**for** x **in** lista:  
 x = x \*\* 2  
 **if** x < 1:  
 menores\_que\_1.append(x)  
*# [0.010000000000000002, 0.09, 0.6400000000000001, 0.010000000000000002]*print(menores\_que\_1)  
  
*# Programação Funcional*menores\_que\_1 = list(map(**lambda** x: x \*\* 2, filter(**lambda** x: x < 1, lista)))  
*# [0.010000000000000002, 0.09, 0.6400000000000001, 0.010000000000000002]*print(menores\_que\_1)

Algumas das funções internas de ordem superior são:

A função Map()

Retorna uma lista dos resultados depois de aplicar a função fornecida a cada item de uma iterável (lista, tupla etc.)

* Sintaxe: mapa (fun, iter)
* Parâmetros:
  + fun: é uma função para a qual o mapa passa cada elemento do dado iterável.
  + iter: é um iterável que deve ser mapeado.
* Tipo de retorno: retorna um iterador da classe de mapa.

*# Return double of n***def** addition(n):  
 **return** n + n  
  
*# We double all numbers using map()*numbers = (1, 2, 3, 4)  
results = map(addition, numbers)  
  
*# Does not Prints the value  
# <map object at 0x000001F60F49D088>*print(results)  
  
*# For Printing value***for** result **in** results:  
 print(result, end=**" "**) *# 2 4 6 8*

Versão resumida usando lambda. Observe que é importante converter o valor de retorno do mapa () como um objeto de lista. É difícil trabalhar com o objeto de mapa retornado se você espera que ele funcione como uma lista. Primeiro, imprimi-lo não mostra cada um dos itens e, em segundo lugar, você pode iterar apenas uma vez.

lst = [1, 2, 3, 4]  
double\_lst = list(map(**lambda** x: x \* 2, lst))  
  
*# [2, 10, 8, 12, 16, 22, 6, 24]*print(**""**)  
print(double\_lst) *# [2, 4, 6, 8]*

A seguir um exemplo comparativo entre imperativa e funcional. Na funcional utilizando lambda percebemos que a implementação não sofre interferência do meio externo (Funções puras), evita loops e sua saída sem o construtor de list é lazy.

*# Gerar uma lista da string # Imperativo*string = **'Python'**lista = [] *# estado inicial***for** l **in** string:  
 lista.append(l) *# cada iteração gera um novo estado*print(lista) *# ['P', 'y', 't', 'h', 'o', 'n']  
  
# Gerar uma lista da string # Funcional*string = **lambda** x: x  
lista = list(map(str, string(**'Python'**))) *# atribuição a um novo objeto*print(lista) *# ['P', 'y', 't', 'h', 'o', 'n']*

A função Filter()

## Filtra a sequência fornecida com a ajuda de uma função que testa cada elemento da sequência como verdadeiro ou não.

## Sintaxe: filtro (fun, sequence)

## Parâmetros:

## fun: uma função que testa se cada elemento de uma sequência é verdadeiro ou não.

## sequence: sequência que precisa ser filtrada, pode ser conjuntos, listas, tuplas ou contêineres de qualquer iterador.

## Tipo de retorno: retorna um iterador que já está filtrado.

**def** fun(variable):  
 letters = [**'a'**, **'e'**, **'i'**, **'o'**, **'u'**]  
 **if** (variable **in** letters):  
 **return True  
 else**:  
 **return False***# sequence*sequence = [**'g'**, **'e'**, **'e'**, **'j'**, **'k'**, **'s'**, **'p'**, **'r'**]  
  
*# using filter function*filtered = filter(fun, sequence)  
  
print(**'The filtered letters are:'**)  
**for** s **in** filtered:  
 print(s) *# e e*

Programa para filtrar apenas os itens pares de uma lista

pares = filter(**lambda** x: x % 2 == 0, range(10))  
*# [0, 2, 4, 6, 8]*print(list(pares))

List Comprehension

List Comprehension é mais eficiente do que usar as funções map() e filter() em relação a processamento e consumo de memória. Em computação, List Comprehension é uma construção que equivale a notação matemática do tipo:

**S = {x^2 ∀ x em ℕ, x>=20}**

Ou seja, S é o conjunto formado por x ao quadrado para todo x no conjunto dos naturais, se x for maior ou igual a 20. Sintaxe:

**lista = [ <expressão> for <referência> in <sequência> if <condição> ]**

Como eventualmente convertemos para listas, devemos reescrever as funções map() e filter() usando a compreensão da lista. Essa é a maneira mais pitônica de escrevê-las, pois estamos aproveitando a sintaxe do Python para fazer listas. Veja como você pode traduzir os exemplos anteriores de map() e filter() para listar compreensões:

values = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
  
*# Map.*add\_10 = list(map(**lambda** x: x + 10, values))  
print(add\_10)  
  
*# Filter.*even = list(filter(**lambda** x: x % 2 == 0, values))  
print(even)  
  
*# Com list comprehensions*add\_10 = [x + 10 **for** x **in** values]  
print(add\_10) *# [11, 12, 13, 14, 15, 16, 17, 18, 19, 20]*even = [x **for** x **in** values **if** x % 2 == 0]  
print(even) *# [2, 4, 6, 8, 10]*

Generator Expressions

Em Python, funções geradoras ou apenas geradores retornam objetos geradores. Esses geradores são funções que contêm a palavra reservada yield. Ao invés de ter que escrever cada gerador com o método \_\_iter\_\_ e next, que é bastante complicado, Python fornece a palavra reservada yield que provê uma maneira fácil para definir geradores. Por exemplo, o iterador de Fibonacci pode ser remodelado como um gerador usando a palavra reservada yield, como mostrado abaixo:

**def** fib(max):  
 a, b = 0, 1  
 **while** a < max:  
 **yield** a  
 a, b = b, a + b  
  
*# 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987***for** n **in** fib(1000):  
 print(n, end=**' '**)

O uso da palavra reservada yield simplifica muito a criação do gerador.

O tipo de dado retornado por List Comprehension e Generator Expression é diferente. Generator Expression usa menos recursos do que o List Comprehension, pois os itens são gerados um de cada vez, apenas quando necessário, economizando principalmente memória. A sintaxe e o conceito são semelhantes aos de uma List Comprehension. Em termos de sintaxe, a única diferença é que você usa parênteses em vez de colchetes.

Para gerar uma lista com o quadrado dos números de 0 e 10 usando list comprehensions é feita da seguinte forma:

squares = [i \*\* 2 **for** i **in** range(10)]  
*# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]*print(squares)

Nós poderíamos usar uma expressão geradora, tal como mostrada abaixo, no lugar de uma list comprehension, e poderíamos então acessar os valores do gerador usando um loop for ou o método next, como mostrado abaixo. Cada valor é computado sob demanda, ou seja, só quando requisitado.

*# squares = <generator object <genexpr> at 0x000001E5BA502A48>*squares = (i \*\* 2 **for** i **in** range(10))  
**for** item **in** squares:  
 print(item, end=**' '**) *# 0 1 4 9 16 25 36 49 64 81*

Também é possível criar geradores através de expressões. Sua sintaxe é bastante parecida com a das list comprehensions, exceto pelo uso dos parênteses ao invés dos colchetes. E que apesar de definidas de um jeito diferente, funcionam da mesma maneira que as funções geradoras.

*# pares = <generator object <genexpr> at 0x000001C7B1E74B48>*pares = (i **for** i **in** range(20) **if not** i % 2)  
**for** item **in** pares:  
 print(item, end=**' '**) *# 0 2 4 6 8 10 12 14 16 18*

No exemplo abaixo, uma sequência mais elaborada de funções geradoras:

**def** numbers():  
 number = 0  
 **while True**:  
 **yield** number  
 number += 1  
  
**def** condition(generator, func=**lambda** x: **True**):  
 **while True**:  
 value = next(generator)  
 **if** func(value):  
 **yield** value  
  
**def** take(n, generator):  
 **for** index **in** range(n):  
 **yield** next(generator)  
  
**def** sum\_values(generator):  
 s = 0  
 **for** value **in** generator:  
 s += value  
 **return** s  
  
result = sum\_values(take(10, condition(numbers(), **lambda** x: x \*\* 2 % 5 == 0)))  
  
print(**"Soma de 10 números natural, em que o quadrado é dividido por 5:"**, result) *# 275*

A função Reduce()

A função reduce() é definida no módulo functools. Como as funções de mapa e filtro, a função reduce() recebe dois argumentos, uma função e um iterável. No entanto, ele não retorna outro iterável, mas retorna um único valor.

A função de argumento é aplicada cumulativamente aos argumentos da lista da esquerda para a direita. O resultado da função na primeira chamada se torna o primeiro argumento e o terceiro item da lista se torna o segundo. Isso é repetido até que a lista se esgote.

No exemplo abaixo, a função mult() é definida para retornar o produto de dois números. Esta função é usada na função reduce() junto com um intervalo de números entre 1 e 4, que são 1,2 e 3. A saída é um valor fatorial de 3.

**import** functools  
  
**def** mult(x, y):  
 print(**"x="**, x, **" y="**, y)  
 **return** x \* y  
  
fact = functools.reduce(mult, range(1, 4))  
print(**'Factorial of 3: '**, fact)  
**"""  
x= 1 y= 2  
x= 2 y= 3  
Factorial of 3: 6  
"""**

Uma versão mais simplificada do cálculo do fatorial de n seria:

**def** fat(n):  
 **return** reduce(**lambda** x, y: x \* y, range(1, n))  
  
*# Saída: 720*print(fat(7))

A função reduce() não é mais builtin no python 3, por isso precisa utilizar a instrução import como abaixo:

from functools import reduce

**from** functools **import** reduce  
  
**def** soma(x1, x2):  
 **return** x1 + x2  
  
*# Saída: 10*print(reduce(soma, [1, 2, 3, 4]))

Uma observação interessante a ser feita é que você não precisa operar no segundo valor na expressão lambda. Por exemplo, você pode escrever uma função que sempre retorna o primeiro valor de um iterável:

values = [1, 2, 3, 4, 5]  
  
*# By convention, we add `\_` as a placeholder for an input  
# we do not use.*first\_value = reduce(**lambda** a, \_: a, values)  
print(first\_value) *# 1*

O método Closure

Basicamente, o método de vincular dados a uma função sem realmente transmiti-los como parâmetros é chamado de closure. É um objeto de função que lembra valores nos escopos anexos, mesmo que eles não estejam presentes na memória. Antes de mergulhar em closure, existem alguns conceitos com os quais devemos estar familiarizados:

1. Funções aninhadas
2. Variáveis não locais.

Uma função definida dentro de outra função é chamada simplesmente de função aninhada. Vamos dar um exemplo de função aninhada e ilustrar o escopo de variáveis não locais.

**def** function\_outer\_enclosing(msg):  
  
 **def** function\_inside\_nested():  
 print (msg)  
   
 function\_inside\_nested()  
  
function\_outer\_enclosing(**'Hello'**) *# Hello*

Podemos ver que a função aninhada printer() conseguiu acessar a variável não local msg da função enclosing.

Observe no exemplo acima, não há variável local msg dentro de function\_inside (), ainda assim, ele imprime a mensagem definida fora dessa função. Isso ocorre porque, quando uma função não encontra uma variável local, ela procura uma variável local definida dentro da função em que está fechada ou aninhada. Isso é chamado de escopo de enclosing.

Vamos considerar o exemplo a seguir para obter informações sobre variáveis não locais e sua importância.

**def** function\_outside():  
 msg = **'Hi'  
 def** function\_inside():  
 msg = **'Hello'** print (msg) *# Hello* function\_inside()  
 print (msg) *# Hi*function\_outside()

Como você pode ver no exemplo acima, quando o function\_outside() é chamado, primeiro o function\_inside() é chamado, imprimindo a variável msg que contém o valor 'Hello'.

Depois disso, quando imprimimos msg novamente, ele imprime o valor 'Hi'. Isso ocorre porque, assim que function\_inside() termina, a variável definida dentro dela também é destruída. Portanto, a variável local para a função externa é impressa. Agora vamos ver o que acontece quando usamos uma variável não local em function\_inside().

**def** outerFunction():  
 msg = **'Hi'  
 def** innerFunction():  
 **nonlocal** msg  
 msg = **'Hello'** print (msg) *# Hello* innerFunction()  
 print (msg) *# Hello*outerFunction()

Quando declaramos uma variável dentro de uma função aninhada como não-local, seu escopo é estendido além dessa função interna para a função externa na qual ela está aninhada. Portanto, a variável msg dentro da função interna é vinculada à variável msg na função externa, substituindo seu valor, mesmo que a função externa desapareça.

Portanto, esse método de vincular dados a uma função sem realmente transmiti-los como parâmetros é chamado de closure. Esse valor no escopo anexo é lembrado mesmo quando a variável sai do escopo ou a própria função é removida do namespace atual. Tente o seguinte código em um intérprete para ver os resultados reais.

obj = outerFunction() *#binding the function to an object***del** outerFunction *#deleting the outer function  
#this returns error as the function is deleted  
# NameError: name 'outerFunction' is not defined*outerFunction()   
obj()  
**"""  
Hello  
Hello  
"""**

Como você pode ver no exemplo acima, mesmo quando a função externa é excluída, o objeto ainda armazena e vincula a variável msg à função interna aninhada. Ou seja, temos um closure no Python quando uma função aninhada faz referência a um valor em seu escopo. Os critérios que devem ser atendidos para criar o closure no Python estão resumidos nos seguintes pontos.

* Nós devemos ter uma função aninhada (função dentro de uma função).
* A função aninhada deve se referir a um valor definido na função anexa.
* A função envolvente deve retornar a função aninhada.

Beneficios dos closures:

* Fornecem um tipo de ocultação de dados, pois são usados como funções de retorno de chamada.
* Pela razão anterior, ajudam a reduzir o uso de variáveis globais.
* Útil para substituir constantes codificadas
* Provam ser eficientes quando temos poucas funções em nosso código.

Quando existem poucos métodos (um método na maioria dos casos) a serem implementados em uma classe, os closures podem fornecer soluções alternativas e mais elegantes. Mas quando o número de atributos e métodos aumentar, implemente melhor uma classe.

Abaixo estão exemplos de soma e multiplixação simples em que um closure pode ser mais preferível do que definir uma classe e criar objetos.

**def** add\_x(x):  
 **def** adder(y):  
 **return** x + y  
 **return** adder  
  
add\_5 = add\_x(5)  
add\_7 = add\_x(7)  
print(add\_5(10)) *# result 15*print(add\_7(10)) *# result 17***def** make\_multiplier\_of(n):  
 **def** multiplier(x):  
 **return** x \* n  
 **return** multiplier  
  
times3 = make\_multiplier\_of(3)  
times5 = make\_multiplier\_of(5)  
  
*# <function make\_multiplier\_of.<locals>.multiplier at 0x0000020A4CAA6CA8>*print(times3)  
print(times3.\_\_closure\_\_[0].cell\_contents) *# 3*print(times3(9)) *# Output: 27*print(times5(3)) *# Output: 15*print(times5(times3(2))) *# Output: 30*

Os decoradores em Python também usam amplamente de closure. Em uma nota final, é bom ressaltar que os valores incluídos na função de closure podem ser encontrados. Todos os objetos de função têm um atributo \_\_closure\_\_ que retorna uma tupla de objetos de célula, se for uma função closure. Referindo-se ao exemplo acima, sabemos que times3 e times5 são funções de fechamento e podemos recuperar seus vaçores através de cell\_contents.

A função Zip()

Esta função retorna uma lista de tuplas, onde a i-ésima tupla contém o i-ésimo elemento de cada um dos argumentos.

x = [1, 2, 3]  
y = [4, 5, 6]  
**for** t **in** zip(x, y):  
 print(t)  
**"""  
(1, 4)  
(2, 5)  
(3, 6)  
"""**

A função zip() retorna uma sequência de tuplas. Lista com tamanhos diferentes serão emparelhadas e a diferença entre elas será desconsiderada.

lst\_a = [6, 7, 8, 9]  
lst\_b = [1, 2, 3, 4, 5,10,11]  
  
**for** x **in** zip(lst\_a, lst\_b):  
 print(x)  
**"""  
(6, 1)  
(7, 2)  
(8, 3)  
(9, 4)  
"""**

A função zip() aponta para o objeto instanciado. Para vermos a lista precisamos passá-la para a função interna list().

x = [1, 2, 3]  
y = [4, 5, 6]  
print(zip(x, y)) *# <zip object at 0x000001DFC92829C8>*print(list(zip(x, y))) *# [(1, 4), (2, 5), (3, 6)]*

Veja a diferença entre um mesmo código na forma imperativa e na funcional:

*# Gera uma lista da string de forma imperativa*string = **'Python'**lst = [] *# estado inicial***for** l **in** string:  
 lst.append(l) *# cada iteração gera um novo estado*print(lst) *# ['P', 'y', 't', 'h', 'o', 'n']  
  
# Gera uma lista da string de forma funcional*string = **lambda** x: x  
lst = list(map(str, string(**'Python'**))) *# atribuição a um novo objeto*print(lst) *# ['P', 'y', 't', 'h', 'o', 'n']*

Combinando zip e map:

a = (1, 2)  
b = (3, 4)  
combinado = list(map(sum, zip(a, b)))  
print(combinado) *# [4, 6]*combinado = list(map(**lambda** x\_y: sum(x\_y), list(zip(a, b))))  
print(combinado) *# [4, 6] # [4, 6]*

Imagine que temos uma lista de tuplas que representam os resultados da consulta. O primeiro valor é o mês, o segundo é a receita total. Suponha que desejemos obter a receita total do primeiro trimestre. Podemos descompactar (unzip) esses resultados e somar a receita.

results = [  
 (**'January'**, 35423.85,),  
 (**'February'**, 31445.75,),  
 (**'March'**, 38525.22,),  
]  
  
months, revenue = zip(\*results)  
  
print(revenue) *# (35423.85, 31445.75, 38525.22)*print(sum(revenue)) *# 105394.82*

Se passarmos várias iteráveis de diferentes comprimentos, o comportamento padrão é compactar o número de elementos no menor iterável. Observe que os elementos 15-19 de long\_list foram ignorados.

short\_list = list(range(5)) *# [0, 1, 2, 3, 4]*long\_list = list(range(10,20)) *# [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]*result = list(zip(short\_list, long\_list))  
print(result) *# [(0, 10), (1, 11), (2, 12), (3, 13), (4, 14)]*

Se você se importa com os elementos à direita, pode usar itertools.zip\_longest.

**import** itertools  
  
result = list(itertools.zip\_longest(short\_list, long\_list))  
*# [(0, 10), (1, 11), (2, 12), (3, 13), (4, 14), (None, 15), (None, 16), (None, 17), (None, 18), (None, 19)]*print(result)

# Currying (Escovando)

Funções de ordem superior permitem [aplicação parcial](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&pto=nl&rurl=translate.google.com.br&sl=auto&sp=nmt4&tl=pt&u=https://en.m.wikipedia.org/wiki/Partial_application&usg=ALkJrhjSJWgArDsePhTlFzkgIZiUpDXipQ) ou [currying](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&pto=nl&rurl=translate.google.com.br&sl=auto&sp=nmt4&tl=pt&u=https://en.m.wikipedia.org/wiki/Currying&usg=ALkJrhigy-C165yAuTS0Iuhwa3MXWtfXNw) , uma técnica que aplica uma função a seus argumentos um de cada vez, com cada aplicativo retornando uma nova função que aceita o próximo argumento. Ou seja, o Currying tem a capacidade de assumir uma função que aceita n parâmetros e a transforma em uma composição de n funções, cada uma delas tendo 1 parâmetro.

Esta prática economiza linhas de código e facilita a sua reutilização, pois permite que um programador expresse sucintamente, por exemplo, a [função sucessora](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&pto=nl&rurl=translate.google.com.br&sl=auto&sp=nmt4&tl=pt&u=https://en.m.wikipedia.org/wiki/Successor_function&usg=ALkJrhi0susHRcnFrLjO6tQ3FF9BPkt39g) como operador de adição parcialmente aplicado ao [número natural](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&pto=nl&rurl=translate.google.com.br&sl=auto&sp=nmt4&tl=pt&u=https://en.m.wikipedia.org/wiki/Natural_number&usg=ALkJrhibWo0j5FF7mXYtFnpVWSdIrjgKxw) um.

**from** functools **import** partial  
  
*# defining a function that sums 2 numbers*plus = **lambda** a, b: a + b  
plus(3, 5) *# result 8  
  
# curring calling partial function by supplying  
# the first parameters with value 1*plus\_one = partial(plus, 1)  
  
*# functools.partial(<function <lambda> at 0x000001A0CE093798>, 1)*print(plus\_one)  
  
*# I can use the new function as normal*plus\_five = plus\_one(5)   
print(plus\_five) *# result 6*

## Avaliação parcial - Partial evaluation

Em [computação](https://pt.qwe.wiki/wiki/Computing) , avaliação parcial é uma técnica para vários tipos diferentes de [optimização do programa](https://pt.qwe.wiki/wiki/Optimization_(computer_science)) de [especialização](https://pt.qwe.wiki/wiki/Specialization_(logic)) . A aplicação mais simples é produzir novos programas que se executam mais rápido do que os originais ao ser garantido para se comportam da mesma maneira.

Um uso direto de currying é o Partial Functions, onde se você tiver uma função que aceite n parâmetros, poderá gerar a partir dele uma ou mais funções com alguns valores de parâmetros já preenchidos.

Às vezes, queremos usar o comportamento de uma função, mas diminua o número de argumentos necessários. O objetivo é "salvar" uma das entradas e criar uma nova função que padronize o comportamento usando a entrada salva. Suponha que desejássemos escrever uma função que sempre adicionasse 2 a qualquer número:

**def** add\_two(b):  
 **return** 2 + b  
  
print(add\_two(4)) *# 6*

A função add\_two é semelhante à função geral, f (a, b) = a + b, apenas o padrão é um dos argumentos (a = 2). No Python, podemos usar o módulo parcial do pacote functools para definir esses padrões de argumento. O módulo parcial recebe uma função e "congela" qualquer número de args (ou kwargs), iniciando no primeiro argumento, depois retorna uma nova função com as entradas padrão.

**from** functools **import** partial  
  
**def** add(a, b):  
 **return** a + b  
  
add\_two = partial(add, 2)  
add\_ten = partial(add, 10)  
  
print(add\_two(4)) *# 6*print(add\_ten(4)) *# 14*

Parciais podem assumir qualquer função, incluindo as da biblioteca padrão.

*# A partial that grabs IP addresses using  
# the `map` function from the standard library.*extract\_ips = partial(  
 map,  
 **lambda** x: x.split(**' '**)[0]  
)  
  
arquivo = open(**'nome\_errado.txt'**, **'r'**, encoding=**"utf8"**)  
lines = arquivo.read()  
print(lines) *# Toque Frágil .....*ip\_addresses = list(extract\_ips(lines))  
print(ip\_addresses) *# ['T', 'o', 'q', 'u', 'e', '', 'F', 'r', 'á', 'g', 'i', 'l' ..*

Funções Lambda

No Python, função anônima significa que uma função está sem nome. Como já sabemos, a palavra-chave def é usada para definir as funções normais e a palavra-chave lambda é usada para criar funções anônimas.

Sintaxe: argumentos lambda: expressão

1. Pode ser definida sem um nome (anônimas) por um curto período de tempo.
2. Esta função pode ter qualquer número de argumentos, mas apenas uma expressão, que é avaliada e retornada.
3. É possível usar funções lambda sempre que objetos de função forem necessários.
4. Você precisa ter conhecimento de que as funções lambda são sintaticamente restritas a uma única expressão.
5. Possui vários usos em campos particulares de programação, além de outros tipos de expressões em funções.

*# funções normais***def** double(x):  
 **return** x \* 2  
  
print(double(5)) *# 10  
  
# funções lambda*dobro = **lambda** x: x \* 2  
print(dobro(5)) *# 10*

No programa acima, lambda x: x \* 2 é a função lambda. Onde x é o argumento e x \* 2 é a expressão que é avaliada e retornada. Esta função não tem nome e retorna um objeto de função que é atribuído ao identificador dobro. Outro exemplo abaixo, com emprego de uma lista:

cube = **lambda** x: x \* x \* x  
print(cube(7)) *# 343*L = [1, 3, 2, 4, 5, 6]  
is\_even = [x **for** x **in** L **if** x % 2 == 0]  
  
print(is\_even) *# [2, 4, 6]*

Se não atribuíssemos lambda a um nome de variável, isso seria chamado de função anônima. Essas funções anônimas são extremamente úteis, especialmente ao usá-las como entrada para outra função. Por exemplo, a função sorted() recebe um argumento opcional da key(uma função) que descreve como os itens em uma lista devem ser classificados.

unsorted = [(**'b'**, 6), (**'a'**, 10), (**'d'**, 0), (**'c'**, 4)]  
  
*# Sort on the second tuple value (the integer).*print(sorted(unsorted, key=**lambda** x: x[1]))  
*#[('d', 0), ('c', 4), ('b', 6), ('a', 10)]*

A função Foldr

# O Fold agrupa uma família de funções de ordem superior pertencentes ao domínio de programação funcional. Em um nível alto, folding permite desconstruir ou reduzir dados. As funções de Fold são de diferentes tipos, as duas principais lineares são foldl e foldr.

O Python já possui foldl porque functools.reduce é uma construção foldl. Pense em foldl como "dobra da esquerda":

# Right fold transformation

Uma função foldl pode ser escrita da seguinte forma com um lambda:

**import** functools  
**import** operator  
  
foldl = **lambda** func, acc, xs: functools.reduce(func, xs, acc)  
result = foldl(operator.sub, 0, [1, 2, 3])   
print(result) *# -6*result = foldl(operator.add, **'L'**, [**'1'**, **'2'**, **'3'**])   
print(result) *# L123*

*# FOLDL Function***def** foldl(func, acc, xs):  
 **return** functools.reduce(func, xs, acc)  
  
print(foldl(operator.sub, 0, [1,2,3])) *# -6*print(foldl(operator.add, **'L'**, [**'1'**,**'2'**,**'3'**])) *# 'L123'*

# Pense em foldr como "dobra da direita" :

# Left fold transformation

Baseando-se no terceiro teorema da dualidade dos autores Richard Bird e Philip Wadler, o foldr pode ser criado como um lambda:

**import** functools  
**import** operator  
  
foldr = **lambda** func, acc, xs: functools.reduce(**lambda** x, y: func(y, x), xs[::-1], acc)  
result = foldr(operator.sub, 0, [1, 2, 3])  
print(result) *# 2*result = foldr(operator.add, **'R'**, [**'1'**, **'2'**, **'3'**])  
print(result) *# 123R*

A instrução xs [:: - 1] é a maneira idiomática do Python para retornar o reverso de uma lista. A outra opção, mais legível, é usar a função reversa interna. Lambdas implementadas como acima não costumam levar a uma boa legibilidade de código. O código a seguir, embora mais longo, pode ser indiscutivelmente mais sustentável:

**def** flip(func):  
 @functools.wraps(func)  
 **def** newfunc(x, y):  
 **return** func(y, x)  
 **return** newfunc  
  
**def** foldr(func, acc, xs):  
 **return** functools.reduce(flip(func), reversed(xs), acc)  
  
print(foldr(operator.sub, 0, [1,2,3])) *# 2*print(foldr(operator.add, **'R'**, [**'1'**,**'2'**,**'3'**])) *# '123R'*

O conceito folding abre as portas para criar muitas outras funções. Ele permite que isso seja feito sem a necessidade de escrever código recursivo explícito ou gerenciar loops. Por exemplo, max, min, sum, prod, any, all, map, filter entre outros, podem ser todos definidos com funções de fold/reduce. Aqui estão alguns exemplos simplistas, usando lambdas por concisão:

**import** functools  
**import** operator  
  
lmax = **lambda** xs: functools.reduce(**lambda** x, y: x **if** x > y **else** y, xs)  
print(lmax([1, 2, 3, 4, 5])) *# 5*lmin = **lambda** xs: functools.reduce(**lambda** x, y: x **if** x < y **else** y, xs)  
print(lmin([1, 2, 3, 4, 5])) *# 1*lsum = **lambda** xs: functools.reduce(operator.add, xs)  
print(lsum([1, 2, 3, 4, 5])) *# 15*product = **lambda** xs: functools.reduce(operator.mul, xs)  
print(product([1, 2, 3, 4, 5])) *# 120*lany = **lambda** pred, xs: functools.reduce(**lambda** x, y: x **or** pred(y), xs, **False**)  
result = lany(**lambda** x: x > 3, [1, 2])  
print(result) *# False*result = lany(**lambda** x: x > 3, [1, 2, 3, 4, 5, 6])  
print(result) *# True*lall = **lambda** pred, xs: functools.reduce(**lambda** x, y: x **and** pred(y), xs, **True**)  
result = lall(**lambda** x: x > 3, [4, 5, 6, 7])  
print(result) *# True*result = lall(**lambda** x: x > 3, [1, 2])  
print(result) *# False*lmap = **lambda** func, xs: functools.reduce(**lambda** x, y: x + [func(y)], xs, [])  
result = lmap(**lambda** x: x + 2, [1, 2, 3, 4, 5])  
print(result) *# [3, 4, 5, 6, 7]*lfilter = **lambda** func, xs: functools.reduce(**lambda** x, y: x + [y] **if** func(y) **else** x, xs, [])  
result = lfilter(**lambda** x: x % 2 == 0, [1, 2, 3, 4, 5, 6, 7, 8, 9])  
print(result) *# [2, 4, 6, 8]*

Abaixo está um cenário simples que pode demonstrar um uso decente de foldl e foldr em Python. Imagine que, dada uma lista, precisamos identificar o último e/ou o primeiro elemento que satisfaça um determinado predicado. o primeiro e o último não exigem nenhum loop ou recursão explícita. O primeiro usa foldr aproveitando a dobra à direita, enquanto o último depende da foldl. Isso pode ser escrito da seguinte maneira:

**import** functools  
  
**def** foldl(func, acc, xs):  
 **return** functools.reduce(func, xs, acc)  
  
**def** flip(func):  
 @functools.wraps(func)  
 **def** newfunc(x, y):  
 **return** func(y, x)  
  
 **return** newfunc  
  
**def** foldr(func, acc, xs):  
 **return** functools.reduce(flip(func), reversed(xs), acc)  
  
**def** first(func, acc, xs):  
 **return** foldr(**lambda** x, y: x **if** func(x) **else** y, acc, xs)  
  
**def** last(func, acc, xs):  
 **return** foldl(**lambda** x, y: y **if** func(y) **else** x, acc, xs)  
  
print(last(**lambda** x: x < 8, 99, [1, 2, 3, 4, 5, 6, 7, 8, 9])) *# => 7*print(first(**lambda** x: x > 3, 99, [1, 2, 3, 4, 5, 6, 7, 8, 9])) *# => 4*print(first(**lambda** x: x > 20, 99, [1, 2, 3, 4, 5, 6, 7, 8, 9])) *# => 99*

### Avaliação de curto-circuito - Short-circuit evaluation

A avaliação de curto-circuito, avaliação mínima, ou avaliação McCarthy é a semântica de alguns [operadores booleanos](https://pt.qwe.wiki/wiki/Logical_connective) em algumas [linguagens de programação](https://pt.qwe.wiki/wiki/Programming_language) em que o segundo argumento é executado, ou apenas avaliadas se o primeiro argumento não é suficiente para determinar o valor da expressão. Ou seja, ao avaliar expressões booleanas como AND e OR, você pode parar assim que encontrar a primeira condição que satisfaça ou negue a expressão.

Curto-circuito explicado pela documentação oficial:

|  |  |  |
| --- | --- | --- |
| **Operação** | **Resultado** | **Descrição** |
| x OR y | Se x é falso, then y else x | Avalia o segundo argumento (y) apenas se o primeiro for false. Ou seja, quando o primeiro argumento avalia a true, o valor total deve ser true. |
| x AND y | Se x é falso, then x else y | Avalia o segundo argumento (y) apenas se o primeiro (x) for True. Ou seja, quando o primeiro argumento avalia a false, o valor total deve ser false |
| NOT x | Se x for falso, then True, caso contrário False | NOT tem uma prioridade mais baixa que os operadores não booleanos. Portanto, NOT a == b é interpretado como NOT (a == b) e a == NOT b é um erro de sintaxe. |

print(**'x'** == (**'x' or 'y'**)) *# True*print(**'y'** == (**'x' or 'y'**)) *# False*print(**'x'** == (**'x' and 'y'**)) *# False*print(**'y'** == (**'x' and 'y'**)) *# True*

Em algumas linguagens de programação ([Lisp](https://pt.qwe.wiki/wiki/Lisp_(programming_language)) , [Perl](https://pt.qwe.wiki/wiki/Perl) , [Haskell](https://pt.qwe.wiki/wiki/Haskell_(programming_language))), por causa da [avaliação preguiçosa](https://pt.qwe.wiki/wiki/Lazy_evaluation), os operadores booleanos usuais são de curto-circuito. Em outros ([Ada](https://pt.qwe.wiki/wiki/Ada_(programming_language)) , [Java](https://pt.qwe.wiki/wiki/Java_(programming_language))), tanto de curto-circuito e operadores booleanos padrão estão disponíveis. Para algumas operações booleanas, como [exclusivo ou](https://pt.qwe.wiki/wiki/Exclusive_or) (XOR), não é possível curto-circuito, porque os operadores são sempre necessários para determinar o resultado.

Geralmente, podemos usar a avaliação de curto-circuito para evitar erros. Um caso comum é a verificação de um valor que é 0 antes de executar um cálculo:

n = 0  
m = 2  
**if** m < 9/n:  
 print(**'m < 1/n'**) *# ZeroDivisionError: division by zero*

Dividir por zero é um erro; portanto, nosso código emite uma exceção e deixa de funcionar. Uma maneira rápida de evitar isso é usar curto-circuito, com o operador AND:

n = 0  
m = 2  
**if** n **and** m < 9/n:  
 print(**'m < 1/n'**)

Você pode esperar que o valor de uma expressão lógica seja sempre Verdadeiro ou Falso, mas, na verdade, esse não é o caso. Python faz algo mais útil.

a = 0  
b = 2  
c = 3  
x = c **or** a  
print(x) *# 3*

Você pode se surpreender ao vê-lo imprimir 3. O que realmente acontece é que o Python imprime o último valor que avalia. Nesse caso, o Python avalia c, que possui um valor 3. Como isso conta como verdadeiro, ele não precisa avaliar a, portanto, ele retorna o valor 3.

Aqui estão mais alguns exemplos, usando os mesmos valores de a, b e c:

x = a **or** b  
print(x) *# 2*x = b **or** c  
print(x) *# 2*x = 0 **or** a  
print(x) *# 0 (the result is false because 0 and a are both false)*x = a **or** b+c **or** b+a  
print(x) *# 5 (b+c, the first true value)*

Aqui está um exemplo da vida real:

username=**''**username = username **or** input(**'Enter username'**)

Nesse caso, a instrução ou avalia primeiro o nome de usuário, que é uma cadeia vazia, e conta como falso. Em seguida, ele avalia a instrução de entrada - ou seja, chama a função de entrada, aguarda o usuário digitar seu nome e retorna esse valor.

Se executássemos essa instrução novamente, mais tarde, quando o nome de usuário tiver um valor, a expressão ou simplesmente retornaria o valor atual e não chamaria input ().

As funções embutidas all () e any () no python também suportam curto-circuito. O exemplo abaixo forneceria uma visão clara de como funciona.

**def** check(i):  
 print (**"geeks"**)  
 **return** i  
  
*# using all()  
# stops execution when false occurs tells the compiler that even if one is  
# false, all cannot be true, hence stop execution further. prints 3 "geeks"*print (all(check(i) **for** i **in** [1, 1, 0, 0, 3]))  
**"""  
geeks  
geeks  
geeks  
False  
"""**print(**"\r"**)  
  
*# using any()  
# stops execution when true occurs tells the compiler that even if one is  
# true, expression is true, hence stop execution further. r prints 4 "geeks"*print (any(check(i) **for** i **in** [0, 0, 0, 1, 3]))  
**"""  
geeks  
geeks  
geeks  
geeks  
True  
"""**

12.3 Jogos

A programação de jogos e uma das áreas mais recompensadoras e inspiradoras. Além disso, o que você aprende programando jogos serve para solucionar diversos outros problemas do dia a dia. Python apresenta diversas bibliotecas externas prontas para a criação de jogos em 2D ou 3D. Essas bibliotecas incluem a manipulação de imagens, sons, controle do tempo e mesmo de arquivos.

* **Pygame (**[**https://www.pygame.org/**](https://www.pygame.org/)**)**

Biblioteca multimídia fácil de aprender. Permite a manipulação de superfícies de desenho e a criaçao de jogos com gráficos em 2D e som. Se você tiver curiosidade de saber como criar jogos simples, visite o projeto Invasores (https://github.com/lskbr/invasores). O projeto foi inteiramente desenvolvido em Python e é open source. Você pode baixar e estudar o código-fonte, fazer modificao5es e testar. Tudo escrito em Python.

* **Panda 3D (http://www.panda3d.org)**

Biblioteca gráfica, desenvolvida pelos estúdios Disney, open source e supercompleta. Com Panda 3D você pode criar jogos profissionais, e o melhor de tudo: usando Python. A biblioteca e poderosa e complexa, tendo sido utilizada em jogos comerciais. Não se esqueça de revisar álgebra linear e estudar gráficos tridimensionais antes de se aventurar na documentação (em inglês).

* **PyOgre (http://wiki.ogre3d.org/Pyogre)**

Biblioteca ou kit de desenvolvimento de jogos 3D. A PyOgre e também poderosa e complexa.

Se você começar a estudar essas bibliotecas, encare essa tarefa como um projeto de pelo menos um ano. Essas bibliotecas são grandes e permitem a criação de aplicativo multimídia, além de jogos, claro. O ideal e realizar esse estudo em grupo: visite o site http://www.unidev.com.br.

Banco de dados

Relacional

Depois de ler a documentação do modulo sqlite3, você pode aventurar-se com bibliotecas mais modernas que isolam seus programas do banco de dados em si, tornando a manipulação de seus dados quase transparente.

* SQLAlchemy (<http://www.sqlalchemy.org>);
* Pony (<https://ponyorm.org/>).

Os bancos de dados clássicos que você deve conhecer são o

* PostgreSQL (https:// www.postgresql.org)
* MySQL (https://www.mysql.coni).

Não Relacional

Existem também diversos bancos de dados modernos que não usam SQL, os chamados bancos NoSQL.

* Redis (<https://redis.io>),
* MongoDB (https://www.mongodb.com/ ),
* Cassandra (http://cassandra.apache.org),
* Amazon DynamoDB (https://github.com/pynamodb/PynamoDB).

Sistemas web

Se você deseja aprender a escrever, ou se interessa por sistemas web, estude o Django (http://www.djangoproject.com). O Django estrutura seus sistemas para que se tornem fáceis de escrever e modificar. Além disso, as bibliotecas do Django já realizam as tarefas repetitivas de comunicação com o servidor web e mesmo com o banco de dados.

Outro framework muito utilizado para desenvolver sistemas web e o Flask (https://flask.palletsprojects.com).

Sistemas em cloud computing também podem ser escritos em Python. Um framework muito interessante e o Chalice, desenvolvido pela própria Amazon Web Services (https://github.con/aws/chalice), e capaz de criar funções com AWS Lambda, uma forma de FaaS (Function as a Service).

CherryPy CherryPy45 é um framework para aplicações Web que publica objetos, convertendo URLs em chamadas para os métodos dos objetos publicados. Com o CherryPy, o programa passa a se comportar como um servidor Web, respondendo a requisições GET e POST.

Geração de HTML

Páginas web nada mais são que a combinação de textos e imagens, interligadas por links. Hoje essa definição vai além com a crescente popularidade de vídeo e as chamadas Aplicações de Internet Ricas (Rich Internet Applications), normalmente escritas em Javascript. Como aqui tratamos apenas de introduzir conceitos de programação, vejamos como utilizar o que já sabemos sobre arquivos para gerar home pages ou páginas web simples.

Toda página web é escrita em uma linguagem de marcação chamada HTML (Hypertext Mark-up Language - Linguagem de Marcação de Hipertexto). Primeiro, precisamos entender o que são marcações. Como o formato HTML e definido apenas com texto simples, ou seja, sem caracteres especiais de controle, ele utiliza marcações, que são sequências especiais de texto delimitado pelos caracteres de menor (<) e maior (>). Essas sequências são chamadas de tags e podem iniciar ou finalizar um elemento. O elemento de mais alto nível de um documento HTML é chamado de <html>. Escreveremos nossas páginas web entre as tags <html> e </html>, em que a primeira marca o início do documento; e a segunda, seu fim.

Como páginas web são arquivos texto, podemos criá-las facilmente em Python. Veja o programa a seguir que cria uma página HTML.

Python oferece recursos mais interessantes para trabalhar com strings, como aspas triplas que permitem escrever longos textos mais facilmente. Elas funcionam como as aspas, mas permitem digitar a mensagem em várias linhas.

Outra vantagem e que não precisamos colocar uma barra antes das aspas, pois agora o fim das aspas também e triplo ("""). Vejamos como gerar páginas web a partir de um dicionário e com o uso de aspas triplas. Vamos utilizar as chaves como título das seções, e o valor como conteúdo do parágrafo.

# Geração de una página web a partir de um dicionário

filmes = {

"drama": ["Cidadão Kane", "0 Poderoso Chefão"],

"comédia": ["Tempos Modernos", "American Pie", "Dr. Dolittle"],

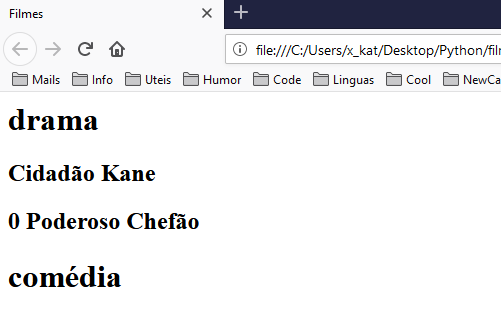
"policial": ["Chuva Negra", "Desejo de Matar", "Difícil de Matar"],

"guerra": ["Rambo", "Platoon", "Tora!Tora!Tora!"]

}

with open("filmes.html", "w", encoding="utf-8") as pagina:

pagina.write("""

 <!D0CTYPE html>

<html lang="pt-BR">

<head>

<meta charset="utf-8">

<title>Filmes</title>

</head>

<body>

""")

for c, v in filmes.items():

pagina.write(f"<h1>{c}</h1>\n")

for e in v:

pagina.write(f"<h2>{e}</h2>\n")

pagina. write( "</body></html>")

Você pode escrever essa página usando um editor de textos. Grave o arquivo com o nome de ola.htnl. Abra em seu browser (navegador) de internet preferido: FireFox, Chrome, Safari, Edge etc. A forma mais fácil de abrir um arquivo em disco no browser e selecionar o menu Arquivo e depois a opção Abrir. Escolha o arquivo pagina.html e veja o que aparece na tela. Se os acentos não aparecerem corretamente, verifique se você gravou o arquivo como texto UTF-8, da mesma forma que fazemos com os programas.

Django

Django é um framework para desenvolvimento web escrito em Python. Ele baseia-se no padrão MTV (Model, Template, View). No ano de 2003, dois Desenvolvedores (Adrian Holovaty e Simon Willison) da World online, uma empresa jornalística no kansas na cidade de Lawrence, abandonaram o PHP e começaram a utilizar o Python para o desenvolvimento de seus sites. Ao construírem sites ricos e iterativos, eles começaram a extrair padrões e estruturas genéricas que permitiam o desenvolvimento mais ágil de aplicações web. Em 21 de julho de 2005, o framework Django foi lançado. A ferramenta teve seu nome inspirado no guitarrista de jazz Django Reinhardt.

Atualmente a ferramenta é bastante estável, várias empresas utilizam-na em seus projetos como, o Mozilla em seu site, a rede social de ideias Pinterest, o Instagram em aplicativo, e a globo em seus sites, globo.com e globoesporte.com o maior site de esportes da américa latina, que recebe mais de 20 milhões de desenvolve grandes aplicações em um tempo significativamente menor.

Django é um framework para web, de alto nível desenvolvido em Python, seguindo as filosofias de desenvolvimento rápido, limpo e com design pragmático. A rapidez no desenvolvimento utilizando Django é modelada de acordo com o princípio DRY - Dont Repeat Yourself, evitando ao máximo códigos duplicados e gerando um ganho de agilidade excelente no ciclo de desenvolvimento web.

Django utiliza MVC e outros padrões, muito bem aceitos por toda a comunidade, em sua filosofia de desenvolvimento, o que leva a fácil manutenção, melhor legibilidade dos códigos e simplicidade. Django é formado pelos principais componentes necessários para um framework web e muito mais, como mapeamento objeto relacional, sistema de administração, sistema de templates, sistema de cache, suporte para internacionalização e mapeamento elegante de urls. Todos os componentes são altamente integrados, já que os componentes não são de terceiros, proporcionando ótima integração entre as partes do framework, com maior coerência e coesão.

Características

* Mapeamento objeto-relacional - é possível definir seus modelos de dados com classes em Python e gerar automaticamente o SQL correspondente e executa-lo no banco de dados a ser utilizado. Mas, se por preciso é possível escrever os comandos SQL também.
* Interface de administração automática - o Django traz automaticamente uma interface de administração automática onde é possível atualizar o conteúdo do seu sistema.
* Urls elegantes - é possível projetar URLs sem nenhuma limitação estrutural.
* Sistema de templates - o django contém um sistema de templates que separa o html do codigo em Python.
* Internacionalização - esse framework tem suporte a aplicações multi-linguagem.
* NewForms - é possível gerar e manipular formulários facilmente através dos modelos de dados definidos.
* Unicode - suporta o unicode, de forma simples

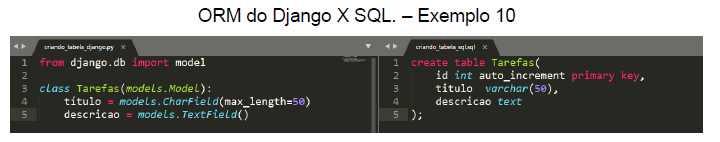
DJANGO ORM

Mapeamento objeto-relacional (ou ORM, do inglês: Object-relational mapping) é uma técnica de desenvolvimento utilizada para reduzir a impedância da programação orientada aos objetos utilizando bancos de dados relacionais. As tabelas do banco de dados são representadas através de classes e os registros de cada tabela são representados como instâncias das classes correspondentes.

Com esta técnica, o programador não precisa se preocupar com os comandos em linguagem SQL; ele irá usar uma interface de programação simples que faz todo o trabalho de persistência.

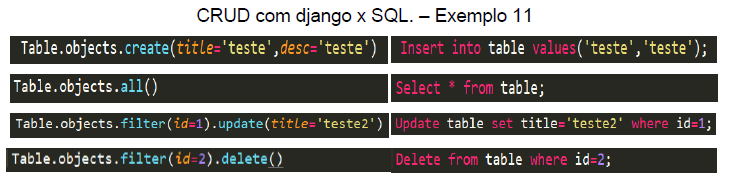
Ao iniciar um projeto Django, a ferramenta já fornece para o analista toda a estrutura básica de arquivos necessários para o desenvolvimento. Nesta estrutura de arquivos é disponibilizado um arquivo para o modelo (models.py), outro para o controle (views.py) um para administração da base de dados (admin.py), outro para a lógica da aplicação (app.py), além de arquivos de configurações para o gerenciamento do projeto (settings.py). Mesmo com esta estrutura de arquivos o desenvolvedor ainda está livre para criar quantos arquivos ele desejar.

O Django possui diversas API’s (Application Programming Interface), e uma delas permite uma maneira diferente de manipulação da base de dados, a API chamada ORM (Object-Relational Mapping - Mapeamento objeto-relacional). Esta API permite que o analista de sistemas manipule todas a operações em banco de dados por códigos python sem a necessidade de scripts SQL. Ao criar uma classe com seus atributos no arquivo (models.py) a API irá espelhar um SQL através desta classe, e criará uma tabela no banco de dados, dispensando o uso de códigos SQL’s.



Neste caso será criado uma tabela chamada tarefas com três campos (titulo,descricao,id). Automaticamente, o Django cria um campo id inteiro como auto incremento para todas as tabelas, sem a necessidade codificar isto, (Caso o desenvolvedor deseje não utilizar este campo criado de forma automática, ele poderá apagá-lo, alterá-lo ou excluí-lo da forma que lhe convém). Também é possível que o desenvolvedor trabalhe com o SQL propriamente dito.

Além de criação de tabelas, também é possível realizar diversos tipos de operações na base de dados, como inserção, atualização, busca, entre outras ações. Abaixo está um exemplo de um pequeno CRUD (Create, Read, Update, Delete) na tabela tarefas.

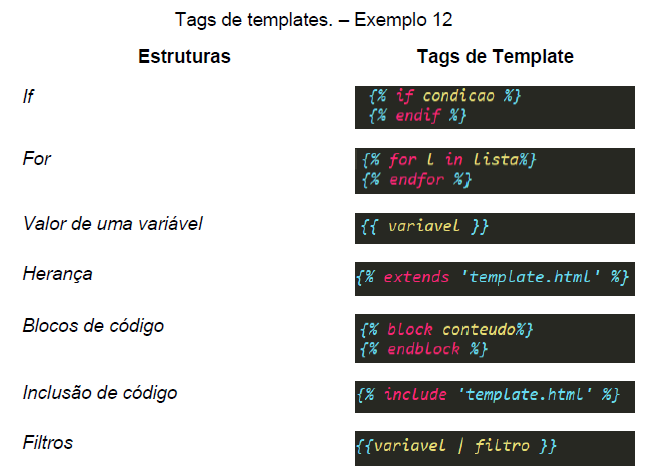


O framework também fornece tabelas criadas automaticamente, dessa forma, não é necessário que o desenvolvedor às crie, porém, a ferramenta permite que ele as altere ou as excluam a qualquer momento. Automaticamente o Django cria uma tabela de usuários, uma tabela de controle de permissões, uma tabela de log, uma para controle de sessões, permissões de usuário, tipo de conteúdo, grupos, permissões de grupos, grupos de usuários, e das migrações do sistema (modificações no banco). Cada uma possui sua finalidade, podendo ser usadas em aplicações de pequeno ou grande porte.

Além do ORM, o Django possui a API admin, que permite que o desenvolvedor administre seu banco de dados sem uso de ferramentas externas, podendo fazer diversas operações ou permitir que o próprio usuário do sistema administre seu banco. Por meio desta API, ele controla a exibição de seus dados, buscas, filtragens, inserções, entre outras ações, tudo isso no próprio sistema em que está sendo desenvolvido.

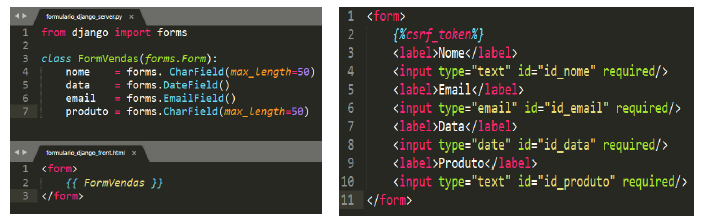
Templates

O framework Django possui um sistema de templates que permite que desenvolvedor crie templates dinâmicos e mais enxutos através de tags de template e filtros. Os templates Django possuem sua própria sintaxe, para separar a linguagem de template da linguagem de servidor. Estas tags também oferecem todo o poder de estruturas condicionais, laços de repetição, inclusões e até mesmo heranças, tornando assim o html mais resumido e dinâmico.

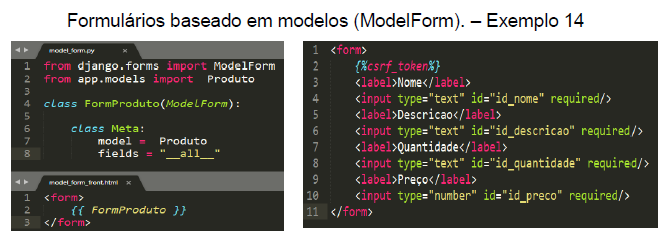


Formulários

Como formulários são características de todo sistema que processe dados de entrada, o Django possui um sistema de formulários onde, formulários html podem ser desenvolvidos diretamente do código Python no servidor. Dessa forma o template html se torna mais limpo e mais rápido para ser desenvolvido, trazendo também mais facilidade no momento da manutenção do sistema.



Outro ponto importante que o framework Django ressalta é que, a maioria dos sistemas, campos em tabelas existentes na base de dados são exatamente os mesmos campos contidos nos formulários, Logo, para agilizar este processo, o Django possui uma classe chamada ModelFom. Para utilizar formulários baseado em modelos, cria-se uma classe que herda de ModelForm e tem como atributo o modelo que será espelhado em um formulário, e os campos a serem exibidos.



O Django já vem com vários formulários pré definidos, bastando apenas importá-los, como formulários de criação de usuários, de login, Alteração de senha, entre outros.

Segurança

A segurança depende muito de como o sistema está sendo implementado, junto a ferramentas extras para a proteção de seus dados. O Django por padrão já fornece ferramentas e técnicas, para um desenvolvimento mais seguro. Dentre elas será citado de modo geral três proteções contra os ataques mais comuns em sites e aplicações web.

Proteção contra falsificação de solicitação de site cruzado (csrf)

Os ataques CSRF( Cross-Site Request Forgery- Falsificação de solicitações entre sites) permite que o usuário mal intencionado, execute ações usando as credenciais de outro usuário, sem que ele saiba. Para se proteger contra ataques desta natureza, o framework fornece um token de autenticação de usuário, que é modificado a cada requisição feita pelo mesmo, fazendo assim com que o usuário seja sempre identificado antes de qualquer tipo de requisição, caso contrário, o servidor retornará um erro HTTP 403 (Proibido), dizendo que o usuário está proibido de realizar requisições.

SQL Inject

Os ataques conhecidos como SQL inject, (injeção de sql) permitem que o usuário mal intencionado, consiga injetar scrípts sql na aplicação, podendo resultar em perda de dados e vazamento de informações. Com a ORM do django (citada acima), estes ataques diminuem drasticamente devido a troca de consultas sql pela codificação da própria API ORM. Ao entrar consultas vindas da ORM o resultado é tratado através do driver do banco de dados, dessa forma não é possível que entre consultas sql cruas dentro do sistema, o que diminui os riscos destes ataques.

Validação do Host no Cabeçalho da Requisição

O Django utiliza o host de cabeçalho para construir urls. Os valores dos hosts são validados para evitar ataques no tipo CSS(Cross Site Scripting) onde outros sites emitem requisição ao seu próprio site. Através desta validação o framework configura os hosts permitidos, e com isso, apenas urls permitidas podem emitir requisições, bloqueando qualquer outro tipo de solicitação externa.

Proteção Contra o Clickjacking

] Clickjacking é o tipo de ataque em que um site malicioso envolve outro site dentro de um frame. Logo ao realizar determinada ação nesse site malicioso, sem que usuário perceba, ele estará efetuando uma ação em outro site. O Django contém a proteção contra o clickjacking através do navegador, onde ele impede que seja processado sites em frames.

Além destes sistemas o django possui diversos outros que defendem o usuário contra vários ataques (Todos estes sistemas de proteção podem ser configurados, otimizados ou desabilitados). Portanto, usando o framework Django não é necessário ter um conhecimento aprofundado na área de segurança, pois, por padrão, a ferramenta implanta diversos métodos de proteção.

Mesmo o desenvolvedor sendo leigo na segurança da informação, ele será capaz de desenvolver um sistema protegido contra os demais tipos de ataques. Vale ressaltar que, toda segurança contém suas limitações, como todo sistema, e que a maior segurança provém do conhecimento de quem está implementando o software.

FLASK

Ciência de dados e inteligência artificial

Para aplicações cientificas ou de cálculo intenso, não se esqueça de olhar os módulos

* NumPy (https://numpy.org/) eScipy (http://www.scipy.org)
* Pandas (https://pandas.pydata.org).

Se você precisar trabalhar com gráficos (coluna, barra, torta, superfície e muitos outros)

* matplotlib (<https://matplotlib.org/>).

Bibliotecas como para machine learning e deep learning

* TensorFlow (<https://www.tensorflow.org/>)
* Keras (<https://keras.io>)
* Scikit-learn (http://scikit-learn.org)

Biblioteca ‘NumPy’

A biblioteca ‘NumPy’ (Numerical Python) estende a linguagem Python com a estrutura de dados ndarray, direcionada para a computação de alto desempenho sobre vetores e matrizes. Ela oferece uma série de benefícios sobre a tradicional lista do Python (abordada no Capítulo III), dentre os quais se destacam a possibilidade de representar e manipular vetores e matrizes de uma forma muito mais natural, o ganho de desempenho na execução de funções matemáticas e estatísticas sobre grandes volumes de dados e a maior facilidade para ler e escrever dados em disco.

A ‘NumPy’ é considerada a “pedra fundamental” da computação científica em Python, não apenas pelas funcionalidades que oferece, mas principalmente pelo fato de as suas propriedades e métodos terem sido utilizados como base para o desenvolvimento de diversas outras bibliotecas importantes para ciência de dados, como a ‘SciPy’ (Scientific Python), ‘pandas’ (Python Data Analysis Library) e ‘Matplotlib’ (Plotting Library).

Este capítulo é inteiramente dedicado à ‘NumPy’. Apesar de, isoladamente, esta biblioteca fornecer poucas funções de alto nível para análise de dados (o “lance” dela é processar vetores e matrizes), há um consenso entre os pythonistas do mundo inteiro a respeito de sua grande importância. Isto porque, ao dominar a computação orientada por arrays da ‘NumPy’, você poderá utilizar outras bibliotecas para ciência de dados com muito mais eficácia. A relação a seguir apresenta os principais tópicos cobertos neste capítulo:

* Propriedades dos ‘ndarrays’;
* Diferentes formas para criar e popular vetores e matrizes;
* Operações de indexação, fatiamento e varredura de ndarrays.
* Aplicação de funções matemáticas e estatísticas sobre ndarrays;

Biblioteca SciPy

SciPy54 é um pacote que expande o NumPy com outras funcionalidades voltadas para a área cientifica. Entre os módulos que fazem parte do pacote, temos:

* linalg: funções de álgebra linear.
* fftpack: transformada de Fourier.
* integrate: funções de integração.
* interpolate: funções de interpolação.
* optimize: funções de optimização.
* signal: processamento de sinais.
* special: funções especiais (Airy, Bessel, etc).

Biblioteca ‘pandas’

A biblioteca ‘pandas’ (Python Data Analysis Library) é, provavelmente, a mais conhecida e utilizada biblioteca para ciência de dados do Python. Ela fornece duas estruturas de dados incrivelmente úteis: Series e DataFrames. A primeira serve para estruturar dados de séries temporais, enquanto a segunda foi especialmente projetada para tornar o processo de manipulação de dados tabulares mais rápido, simples e eficaz. As operações sobre estas EDs consistem basicamente em uma combinação das técnicas para processamento de arrays presentes na 'NumPy' com um conjunto de funções específicas para a manipulação de dados tabulares que se assemelham muito às oferecidas pelo Excel e pela linguagem SQL. Alguns exemplos:

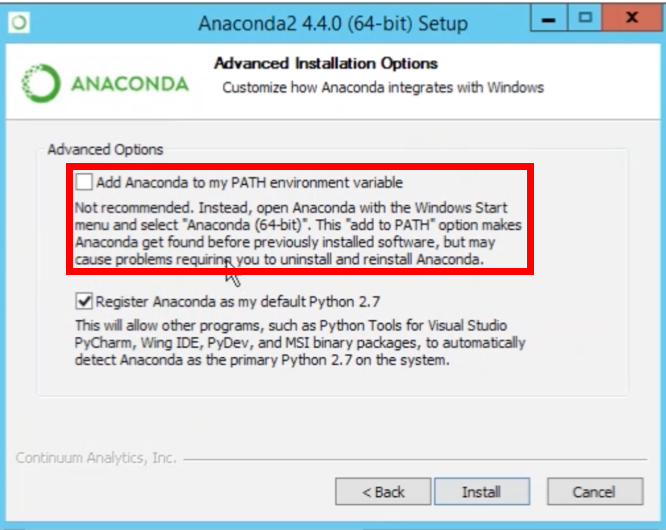
* Importar dados estruturados de diferentes tipos de fontes, como arquivos texto, bancos de dados relacionais, planilhas eletrônicas, etc.
* Combinar, de forma inteligente, observações provenientes de diferentes bases de dados (operação conhecida como merge ou join);
* Produzir resultados agregados e tabulações (group by);
* Limpar e transformar bases de dados (ex.: discretização de variáveis, aplicação de filtros sobre linhas e colunas de tabelas, ordenação dos dados, tratamento de dados ausentes, etc.)

Configurar um ambiente de ciência de dados com Anaconda e PyCharm

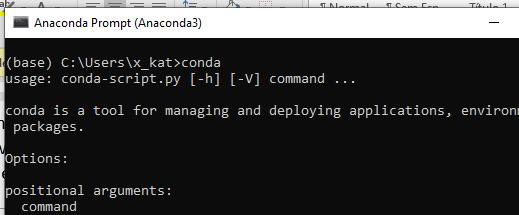
O **Anaconda** (paródia com o nome de outra cobra, a python) é uma iniciativa que tem como objetivo agregar as principais ferramentas para análise de dados no [python](https://www.monolitonimbus.com.br/python/). Para utilizá-lo, baixe seu único arquivo contendo um novo python e todas as bibliotecas, que serão instaladas na home do próprio usuário (sem interferir na instalação principal). O arquivo instalador pode ser baixado do [site do Anaconda](https://www.anaconda.com/download/), escolhendo sistema operacional, versão do python e arquitetura. Depois, torne-o executável (chmod +x nome\_do\_arquivo.sh) e siga os passos.

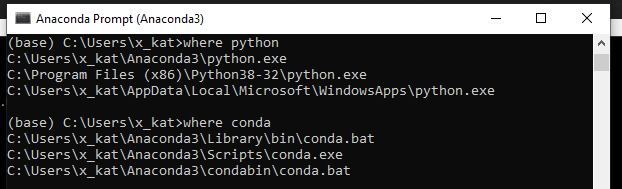
Após instalar o Anaconda, abra uma janela do terminal, digite conda e pressione enter. Se você vir algo semelhante à imagem abaixo, então instalou o Anaconda e o Conda CLI corretamente.

No entanto, no Windows, a variável de ambiente PATH não é mais alterada por padrão, pois isso pode causar problemas com outro software.



A abordagem recomendada é usar o Anaconda Navigator ou o Prompt de Comando do Anaconda (localizado no Menu Iniciar em “Anaconda”) quando você quiser usar o software Anaconda.



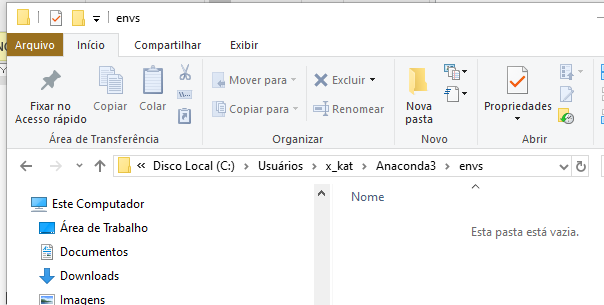


Mas caso, deseje utilizar o prompt do Windows, é necessário adicionar a variáveis do ambiente os seguintes caminhos:

C:\Users\RajaRama\Anaconda3\Scripts

C:\Users\RajaRama\Anaconda3

O caminho padrão para a pasta envs onde todos os seus ambientes criados serão colocados é /Users/<your user name>/anaconda/envs/ .

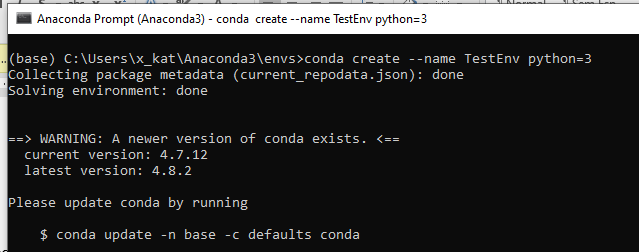


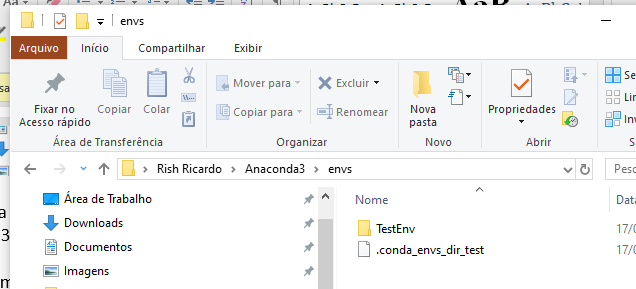
Navegue para a pasta envs antes de criar um novo ambiente. Execute o seguinte comando.

cd C:\Users\x\_kat\Anaconda3\envs

Para criar um novo ambiente, execute o seguinte comando.

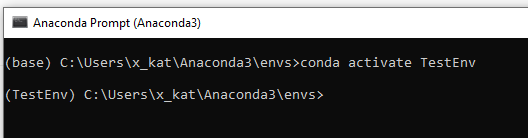
conda create --name TestEnv python=3





Antes de instalarmos novas bibliotecas python no novo ambiente, precisamos ativar o ambiente. Para ativar o ambiente criado, execute o seguinte comando.

conda activate TestEnv



Em seguida, queremos instalar quatro bibliotecas Python de terceiros. Execute o seguinte comando.

conda install numpy pandas scikit-learn matplotlib

Isso deve instalar as quatro bibliotecas numpy, pandas, scikit-learn e matplotlib. Em termos de introdução ao aprendizado de máquina, essas quatro bibliotecas devem oferecer tudo que precisamos.

Quando as bibliotecas estão instaladas, você pode verificar se está tudo bem iniciando um console Python. Isso é realizado executando o seguinte.

python

Depois que o console do Python estiver em execução, execute o seguinte e pressione enter para cada uma das quatro bibliotecas.

>>> import numpy

>>> import pandas

>>> import sklearn

>>> import matplotlib

Se as bibliotecas estiverem instaladas corretamente, você não deverá receber erros.

### Lista de comandos Conda úteis

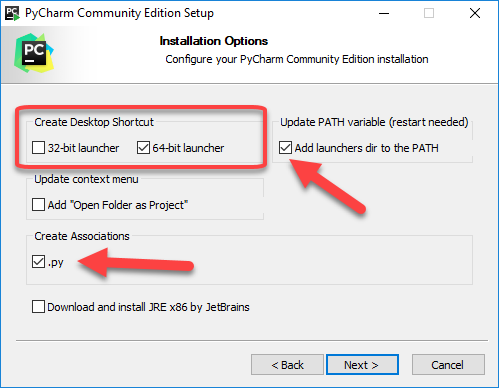
|  |
| --- |
| conda create --name <my environment name> python=<python version> |
| Este comando consiste em duas partes. A primeira parte conda create --name seguida do nome que você deseja fornecer ao seu ambiente cria o ambiente conda. Próxima parte python=3 especifica qual versão do python você deseja instalar no ambiente. Ao executar o conda create --name TestEnv python=2.7 O CLI conda criaria um ambiente chamado TestEnv com o Python 2.7 instalado. |
| source activate <my environment name> |
| Este comando ativa um ambiente específico. Se tivéssemos criado um ambiente chamado TestEnv, poderíamos ativar esse ambiente executando a seguinte source activate TestEnv comando: source activate TestEnv . Este comando é um pouco diferente no Windows. No Windows, você activate TestEnv . |
| source deactivate |
| Este comando desativa o ambiente. Novamente no Windows, é um pouco diferente. No Windows, você escreverá deactivate . |
| conda install <library name> |
| Após ativar um ambiente criado, você provavelmente precisará instalar bibliotecas adicionais, a menos que possa gerenciar com as principais bibliotecas Python, pré-instaladas. No entanto, quando você precisar instalar uma biblioteca de terceiros, você desejará usar acima do comando da Conda CLI. Por exemplo, depois de ativar o ambiente TestEnv, você precisa instalar a biblioteca do pandas, isso é feito executando o conda install pandas . Isso instalará a versão mais recente da biblioteca do pandas disponível. |
| conda update <library name> |
| Se em algum momento você precisar atualizar uma biblioteca que você já instalou, poderá fazer isso usando o comando acima. Por exemplo, você instalou a biblioteca do pandas e, após algum tempo, é necessário atualizar a biblioteca para uma versão mais recente. Isso é feito executando o conda update pandas . Isso atualizará a biblioteca do pandas para a versão mais recente. |
| conda remove --name <my environment name> <library name> |
| Este comando pode ser usado quando você deseja remover uma biblioteca já instalada de um ambiente ativado. Por exemplo, você criou o ambiente TestEnv e ativou o ambiente. Além disso, instalou os pandas da biblioteca no ambiente ativo. Agora você precisa remover os pandas da biblioteca. Isso é feito executando a conda remove --name TestEnv pandas . |
| conda remove --name <my environment name> --all |
| Este comando pode ser usado quando você precisar remover um ambiente criado com suas bibliotecas de terceiros instaladas. Por exemplo, depois de criar o ambiente TestEnv e instalar os pandas da biblioteca. Para remover esse ambiente com suas bibliotecas instaladas, você pode executar conda remove --name TestEnv --all . |
| conda update |
| Se você precisar atualizar toda a instalação do Anaconda para a versão mais recente, execute o comando acima. |

### Criando um projeto no PyCharm

Agora você instalou o Anaconda e criou um ambiente. Agora queremos usar o PyCharm para criar um projeto e executar o código Python usando o ambiente criado. Aqui, presumo que você instalou o PyCharm.

Ao instalar, certifique-se de:

* Selecione o iniciador de bits apropriado para sua máquina
* Selecione a opção para criar associações aos arquivos ".py"
* Selecione a opção para adicionar o diretório dos lançadores ao PATH



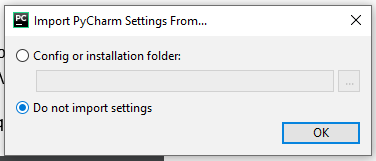
A seleção da opção para adicionar o diretório dos lançadores ao PATH é muito importante. Isso exigirá que você reinicie o seu PC para concluir a instalação.

Primeiro, queremos criar uma pasta que será a nossa pasta do projeto. Você pode criar a pasta onde quiser na sua máquina.

Crie uma pasta de projeto, por exemplo, chamada ProjectTestEnv.

C:\Users\x\_kat\Documents\Projetos\ProjectTestEnv

Agora abra o PyCharm e clique em Abrir.



Ambiente virtual para o R

Caso queira usar o [R](https://www.monolitonimbus.com.br/linguagem-r/), o indicado é criar um outro ambiente virtual só para ele. Depois, use o canal “conda-forge” para instalar o R, conforme segue:

# Criar ambiente virtual para o R

conda create -n renv

# Ativar ambiente virtual

conda activate renv

# Instalar o R no novo ambiente virtual

conda install -c conda-forge R

Para instalar **pacotes** do R no ambiente virtual, você pode usar o seguinte comando, substituindo PACKAGENAME pelo nome do pacote R que aparece no CRAN (Comprehensive R Archive Network):

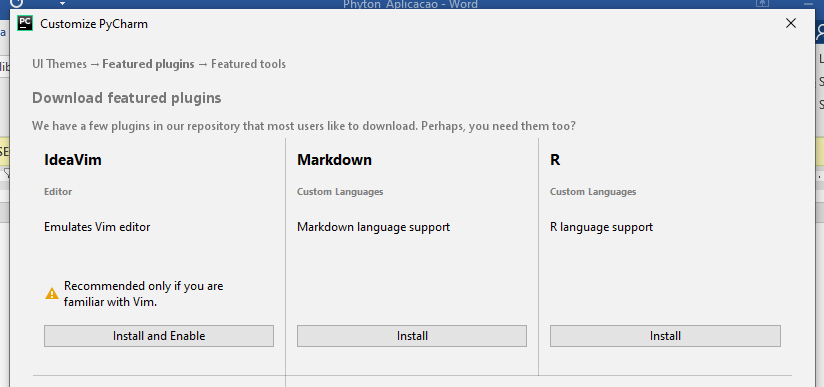
$ conda search -f r-PACKAGENAME

Os [pacotes R disponíveis no conda](https://docs.anaconda.com/anaconda/packages/r-language-pkg-docs/) estão listados no link. Caso não seja possível instalar o pacote através desse comando, entre no terminal R (certifique-se de estar no ambiente virtual dele) e faça a instalação especificando o caminho. Veja esse exemplo, considerando que o R está no ambiente virtual “renv”:

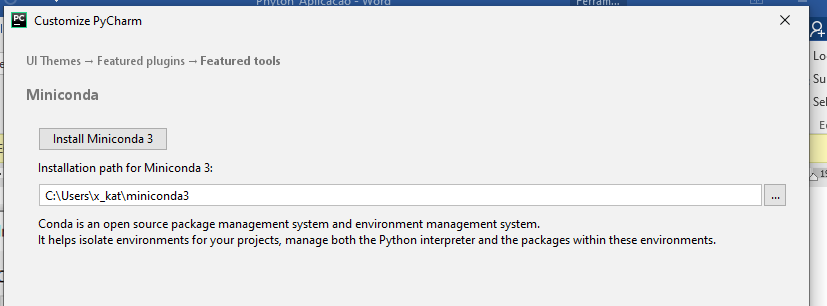
> install.packages("PACKAGENAME", dependencies=TRUE, "~/miniconda3/envs/renv/lib/R/library/")

Para o caso do ambiente virtual ser o “base”, o caminho das bibliotecas R é “/home/user/anaconda3/lib/R/library”.

Se tiver algum shell script que chame scripts em R ou em python, deve-se incluir o comando de ativar o respectivo ambiente virtual antes de sua execução.



**Miniconda** contêm apenas o gerenciador de pacotes (**conda**) e o Python. Então, você pode usar o comando conda para instalar quaisquer outros pacotes e criar ambientes, etc. Existem duas variantes do instalador: Miniconda é baseado em Python 2 e Miniconda3 é baseado em Python 3. Observe que a escolha de qual Miniconda está instalado afeta apenas o ambiente raiz. Independentemente de qual versão do Miniconda você instalar, você ainda pode instalar os ambientes Python 2.x e Python 3.x.



Quando o PyCharm é aberto, você deve ver algo como a imagem abaixo.

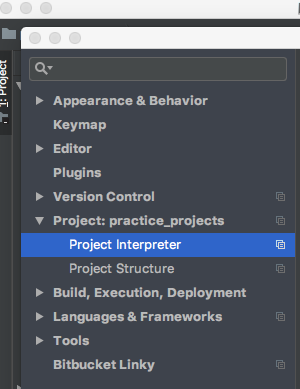


Navegue até a pasta do projeto criada recentemente e clique em Abrir. O PyCharm agora iniciará um novo projeto.

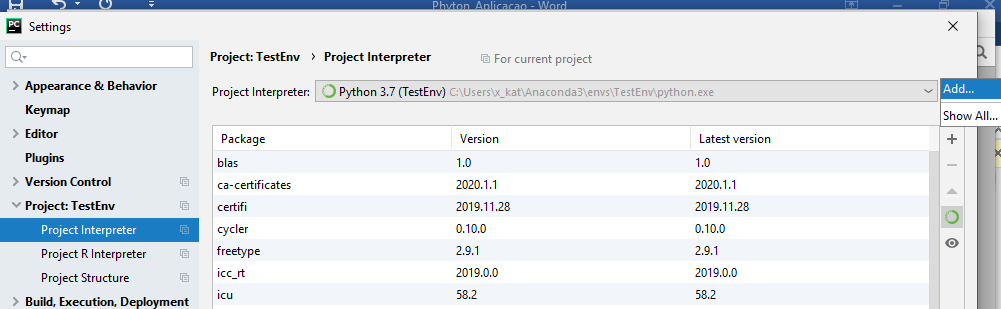
### Configurando o intérprete para o nosso projeto no PyCharm

Agora, toda vez que executamos o código do nosso Projeto Pycharm, queremos que o código use nosso ambiente Conda recém-criado, onde já instalamos as bibliotecas que queremos usar.

Quando o PyCharm terminar, inicie o novo projeto, navegue para “File” -> “Settings”.

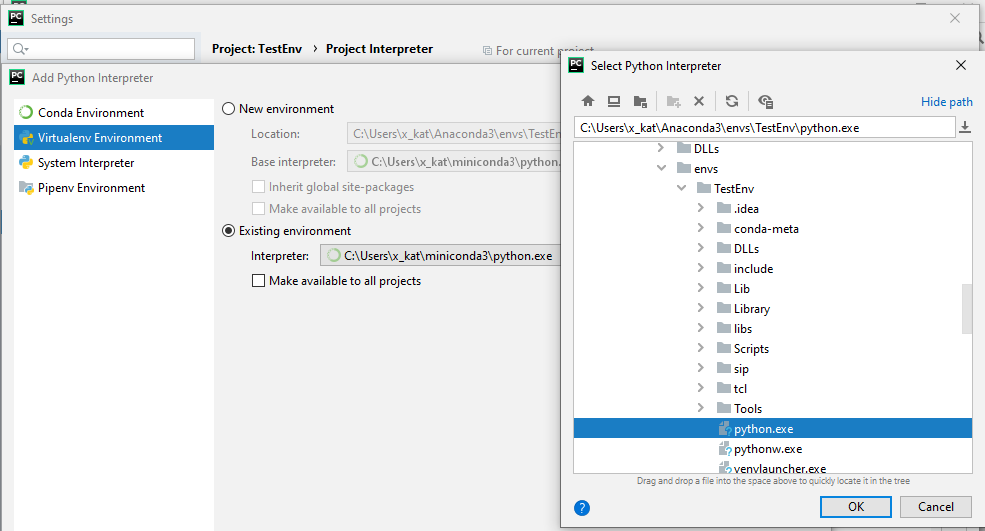


Uma nova janela se abrirá. Selecione "Intérprete" e clique no ícone no canto superior direito, depois clique novo intérprete em "Adicionar ..."



Uma nova janela se abrirá. Aqui, selecione “Ambiente virtual”, clique em “Ambiente existente” e clique no ícone que contém “…”. Uma nova janela se abrirá.

Aqui você precisa navegar para o ambiente criado, por C:\Users\x\_kat\Anaconda3\envs\TestEnv\python.exe



Agora pressione “OK” para todas as três janelas abertas. O PyCharm agora deve definir o ambiente criado como seu intérprete de projeto.

Isso é tudo. Agora você pode usar o PyCharm para criar arquivos python na pasta do projeto, clicando com o botão direito do mouse na visão geral da pasta e adicionando novos arquivos python. Você pode executar qualquer arquivo python na pasta do projeto clicando com o botão direito do mouse no arquivo e selecionando “Executar '<nome do seu arquivo>'”

Configurando o intérprete Método Prandiano

Para configurar o diretorio-padrao do PyCharm, seguir o algoritmo abaixo:

1. Acessar o "PyCharm";
2. Clicar com o botão esquerdo do mouse no botão "File" na barra de ferramentas;
3. Clicar com o botão esquerdo do mouse no botão "Settings...";
4. Expandir o campo "Build, Execution, Deployment";
5. Expandir o campo "Console";
6. Clicar com o botão esquerdo do mouse no botão "Python Console";
7. No campo "Working directory:" selecionar o caminho "[C:\CPP](file://C:/CPP)";
8. Clicar com o botão esquerdo do mouse no botão "OK".

Para configurar o interpretador Anaconda no PyCharm, seguir o algoritmo abaixo:

1. Apos instalar o Anaconda;
2. Executar o "PyCharm";
3. Clicar com o botão esquerdo do mouse no botão "File" na barra de ferramentas
4. Clicar com o botão esquerdo do mouse no botão "Default Settings...";
5. Expandir o campo "Project";
6. Clicar com o botão esquerdo do mouse no botão "Project Interpreter";
7. Clicar com o botão esquerdo do mouse no botão "Add Local Python Interpreter";
8. Clicar com o botão esquerdo do mouse no botão "System Interpreter";
9. Na barra de endereço, especificar "[C:\ProgramData\Anaconda3\python.exe](file://C:/ProgramData/Anaconda3/python.exe)";
10. No "Conda Enviroment" selecionar "Make available to all project";
11. Clicar com o botão esquerdo do mouse no botão "OK";
12. Clicar com o botão esquerdo do mouse no botão "OK".

Os comandos básicos do PyCharm:

|  |  |
| --- | --- |
| **Atalho** | **Descrição** |
| Crtl + Alt + S | Configurações |
| Crtl + S | Salvar tudo |
| Alt + Shift + F10 | Executar |
| Alt + Shift + F9 | Depurar |
| Shift + F10 | Executar projeto ativo |
| Shift + F9 | Depurar projeto ativo |
| F8 | Depurar passo-a-passo |
| Shift + F8 | Depurar até o próximo ponto de interrupção |
| F7 | Depurar funções internas |
| Crtl + Alt + L | Endenta o código |

NUMPY(Numerical/Extended Python) - vetores e matrizes

Exemplo: Cálculo de IMC

**def** categoria(imc):  
 categorias = [**"Abaixo do Peso"**, **"Peso Normal(Saudável)"**,  
 **"Acima do peso(Sobrepeso)"**, **"Obesidade Grau I(Gordo)"**,  
 **"Obesidade Grau II(Severa)"**, **"Obesidade Grau III(Mórbido)"**]  
 **if**(imc <= 18.5) :  
 print(categorias[0])  
 **elif** (imc > 18.5 **and** imc < 24.9):  
 print(categorias[1])  
 **elif** (imc > 24.9 **and** imc < 29.9):  
 print(categorias[2])  
 **elif** (imc > 29.9 **and** imc < 34.9):  
 print(categorias[3])  
 **elif** (imc > 34.9 **and** imc < 39.9):  
 print(categorias[4])  
 **else** :  
 print(categorias[5])  
  
**def** peso():  
 massa = float(input(**"Digitar sua massa: "**))  
 altura = float(input(**"Digitar sua altura: "**))  
  
 sexo = input(**"Digitar: M(masculino) ou F(feminino) \n"**)  
 **if** (sexo == **"M"**):  
 imc = massa / altura \*\* 2  
 print(**'IMC: {:.2f}'**.format(imc))  
 categoria(imc)  
 peso\_ideal = (72.2 \* altura) - 57  
 print(**'"O seu peso ideal seria: {:.2f}'**.format(peso\_ideal))  
 **else**:  
 imc = massa / altura \*\* 2  
 print(**"o seu imc é "** + str(imc))  
 resp = input(**"deseja ver o seu peso ideal, digite: sim ou não "**)  
 categoria(imc)  
 peso\_ideal = (62.1 \* altura) - 44.7  
 print(**'"O seu peso ideal seria: {:.2f}'**.format(peso\_ideal))  
peso()

*# Função np.array***import** numpy **as** np  
  
x = np.array([[1, 2], [3, 4]])  
print(x)  
print(type(x))  
y = np.array([[2, 4, 6], [8, 10, 12], [14, 16, 18]])  
print(y)  
print(type(y))  
  
*# Perceber o ponto após os números*z = np.array([[1, 3], [5, 7]], dtype=float)  
print(z)  
print(type(z))

SCIPY (Extended Numpy) - área cientifica

MATPLOTLIB - geração de gráficos

Por se tratar de uma linguagem cientifica, obviamente, o Python possui como recurso pacotes para plotar grancos, sejam estes 2 — D ou 3 — D. Dentre esses pacotes, tem-se o pacote matplotlib que permite plotar gráficos com muita simplicidade.

Matplotlib Existem vários pacotes de terceiros para a geração de gráficos disponíveis para Python, sendo que o mais popular deles é o Pylab / Matplotlib56. O pacote tem dois módulos principais:

* matplotlib: módulo que oferece uma abstração orientada a objetos aos recursos do pacote.
* pylab: módulo que oferece uma coleção de comandos que se assemelha ao Matlab, e é mais adequado para o uso interativo.

Função plt.plot

Retornar a plotagem de um gráfico, a partir dos vectores x e y.

plt.plot(x, y, [color], [marker], [linestyle])

Sendo:

* x = Obrigatorio. Vector domínio/categoria
* y = Obrigatorio. Vector imagem/serie
* [color] = Opcional. Color/cor

|  |  |  |
| --- | --- | --- |
| **Símbolo** |  | **Cor** |
| k | black | preto |
| b | blue | azul |
| c | cyan | ciano |
| g | green | verde |
| m | magenta | magenta |
| r | red | vermelho |
| w | white | branco |

* [marker] = Opcional. Mar/ser/marcador

|  |  |
| --- | --- |
| **Símbolo** | **Marcador** |
| \* | Estrela ou asterisco |
| o | Circulo |
| x | Cruz |
| d | Losango |
| h | Hexágono |
| P | Pentágono |
| . | Ponto |
| + | Sinai de positivo |
| s | Quadrado |
| v | Triangulo voltado para baixo |
| < | Triangulo voltado para a esquerda |
| > | Triangulo voltado para direita |
| ^ | Triangulo voltado para cima |

* [linestyle] = Opcional. Spec/tipo de linha

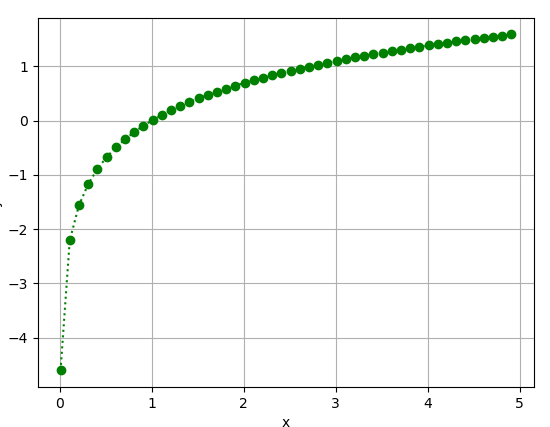
|  |  |
| --- | --- |
| **Simbolo (s)** | **Tipo de linha** |
| **-** | Continua |
| **--** | Tracejada |
| **-.** | Traço-ponto |
| **:** | Pontilhada |

|  |  |  |
| --- | --- | --- |
| **Função** | **Descrição** | **Exemplo** |
| xlabel(str) | Texto do título do eixo x do gráfico | plt.xlabel("Meses") |
| ylabel(str) | Texto do título do eixo y do grafico | plt.ylabel("Qtde") |
| title(str) | Texto do título do gráfico. | plt.title("Quantidade Colaboradores") |
| grid() | Inserir um grid ( grade ) no gráfico | plt.grid() |
| legend(str) | Texto(s) da(s) legenda(s) no grafico. | plt.legend(["f(x) = a",  "g(x) = "]) |
| close() | Fechar janela grafica ativa | plt.close() |
| close(name/num) | Fechar pelo Nome ou número do gráfico a ser fechado. | plt.close("plot\_grafico") |
| close("all") | Fechar todas as janelas graficas. | plt.close("all") |
| savefig(str) | Salva um arquivo de imagem do gráfico | plt.savefig("plot\_graficos.png") |
| show() | Abre o gráfico | plt.show() |

Exemplo Função Logarítmica

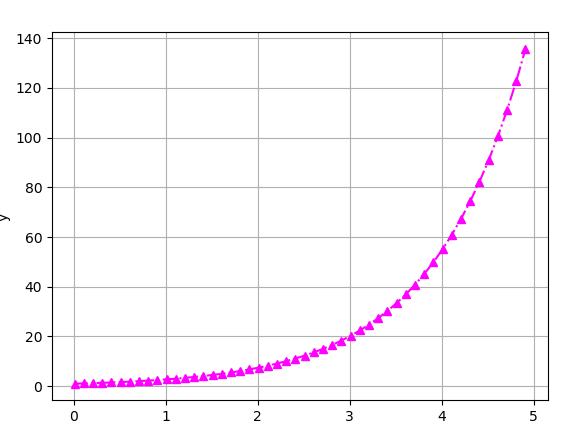
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
  
x = np.arange(0.01, 5, 0.1)  
f = np.log(x)  
plt.plot(x, f,  
 color=**"g"**,  
 marker=**"o"**,  
 linestyle=**":"**)  
plt.xlabel(**"x"**)  
plt.ylabel(**"y"**)  
plt.grid()  
plt.savefig(**"plot\_funcao\_logaritmica\_example.png"**)  
plt.show()





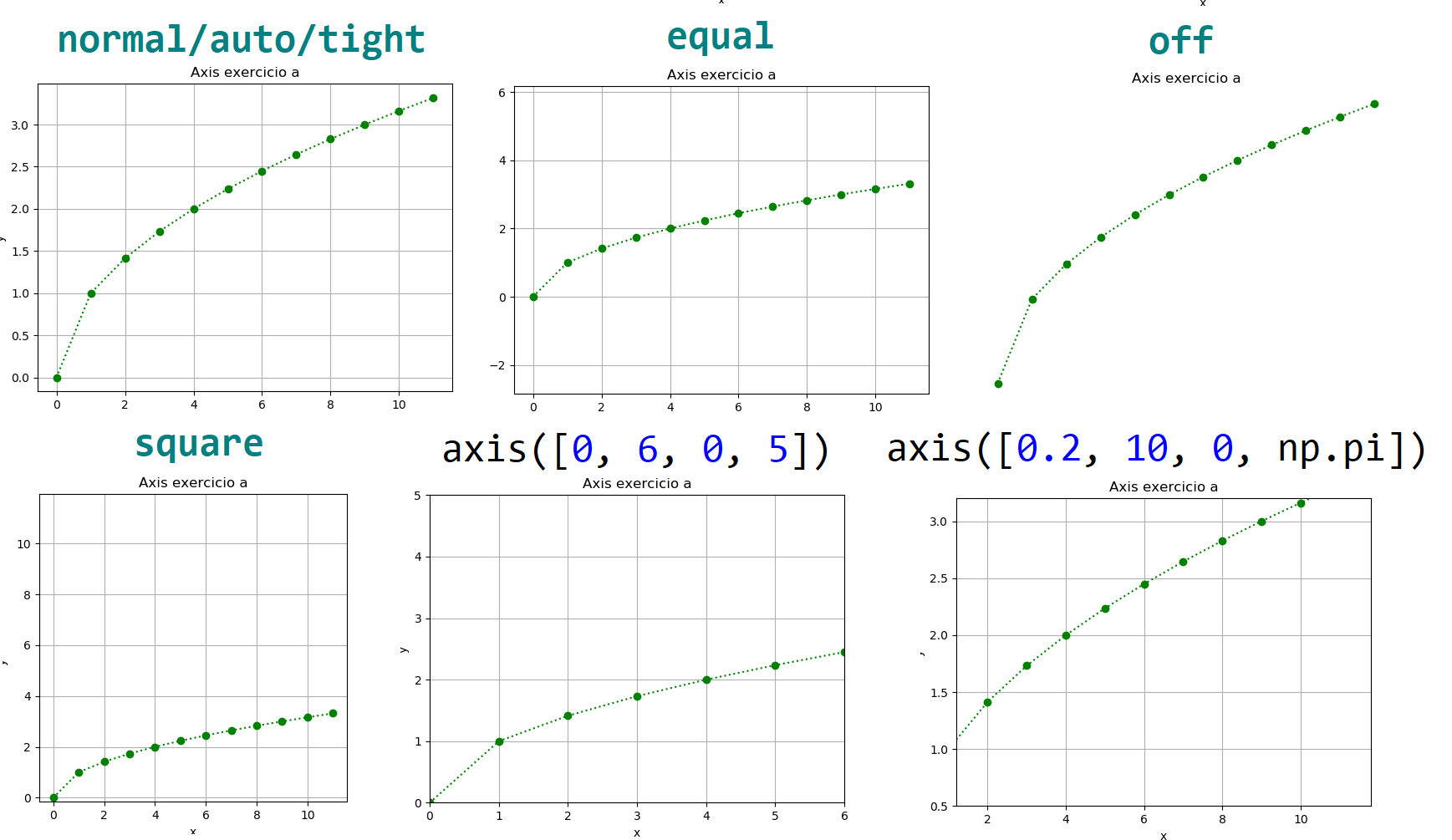
Exemplo Função Exponencial

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
  
x = np.arange(0.01, 5, 0.1)  
f = np.exp(x)  
plt.plot(x, f,  
 color=**"magenta"**,  
 marker=**"^"**,  
 linestyle=**"-."**)  
plt.xlabel(**"x"**)  
plt.ylabel(**"y"**)  
plt.grid()  
plt.savefig(**"plot\_funcao\_exponencial\_example.png"**)  
plt.show()



Personalizar Eixos

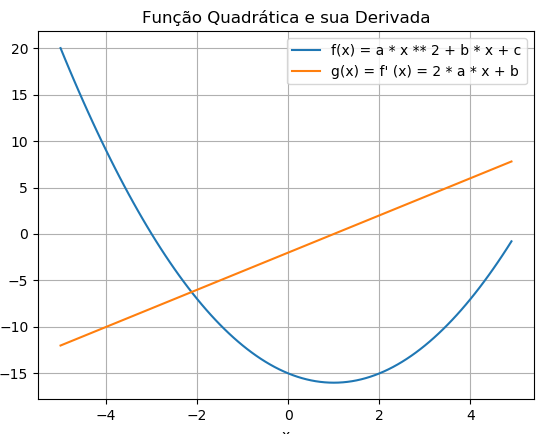
|  |  |  |
| --- | --- | --- |
| **Função** | **Sintaxe** | **Descrição** |
| axis | axis([xmin, xmax, ymin, ymax]) | Personalizar os valores mínimos e máximos dos eixos x e y |
| axis auto | axis("auto") | Retornar os eixos x e y para a escala automática |
| axis equal | axis("equal") | Personalizar a escala dos eixos x e y como sendo iguais |
| axis off | axis("off”) | Limpar personalização dos eixos do gráfico se personalizadas anteriormente |
| axis square | axis(“square”) | Personalizar o gráfico para uma região quadrada |
| axis tight | axis("tight") | Personalizar os valores mínimos e máximos dos eixos x e y a partir dos valores do gráfico |
| axis normal | axis("normal") | Despersonaliza as funções axis square, axis equal e axis tight |



Corriqueiramente é necessário plotar mais de uma série de dados em um mesmo gráfico. Para tanto, pode-se usar a função plt.plot.

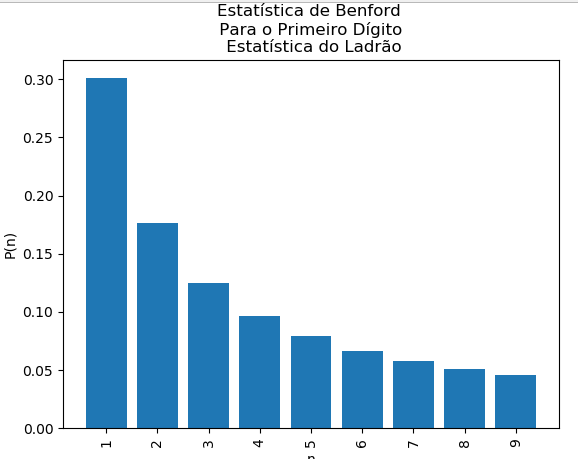
Exemplo Múltiplos Gráficos

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
  
plt.close(**"all"**)  
  
x = np.arange(-5, 5, 0.1)  
a = 1  
b = -2  
c = -15  
  
f = a \* x \*\* 2 + b \* x + c  
g = 2 \* a \* x + b  
  
plt.plot(x, f)  
plt.plot(x, g)  
  
plt.xlabel(**"x"**)  
plt.ylabel(**"y"**)  
plt.title(**"Função Quadrática e sua Derivada"**)  
plt.grid()  
plt.legend([**"f(x) = a \* x \*\* 2 + b \* x + c"**,  
 **"g(x) = f' (x) = 2 \* a \* x + b"**])  
  
plt.savefig(**"multiplos\_graficos\_first\_example.png"**)  
plt.show()



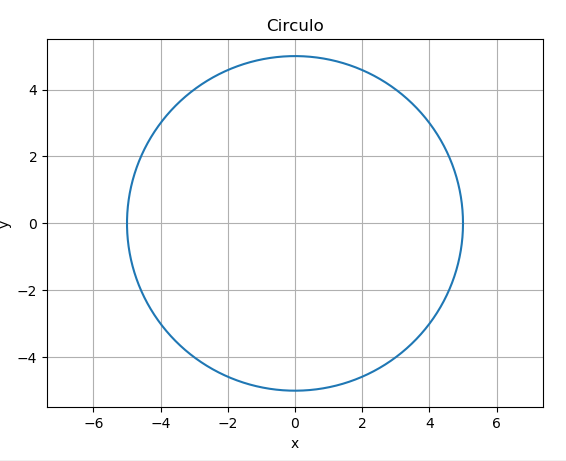
Exemplo Gráfico de Barras

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
  
plt.close(**"all"**)  
  
meses = [**"Jan"**, **"Fev"**, **"Marc"**, **"Abr"**,  
 **"Mai"**, **"Jun"**, **"Jul"**, **"Ago"**,  
 **"Set"**, **"Out"**, **"Nov"**, **"Dez"**]  
  
qtde\_colaboradores = [566, 512, 571, 578,  
 515, 533, 596, 535,  
 548, 592, 574, 576]  
  
valores\_x = np.arange(1, 13, 1)  
  
plt.bar(valores\_x, qtde\_colaboradores)  
  
plt.xlabel(**"Meses"**)  
plt.ylabel(**"Qtde"**)  
plt.title(**"Quantidade Colaboradores"**)  
  
*# Fica ruim com barras  
# plt.grid()*plt.savefig(**"plot\_bar\_first\_example.png"**)  
plt.show()



Exemplo Gráfico Círculo

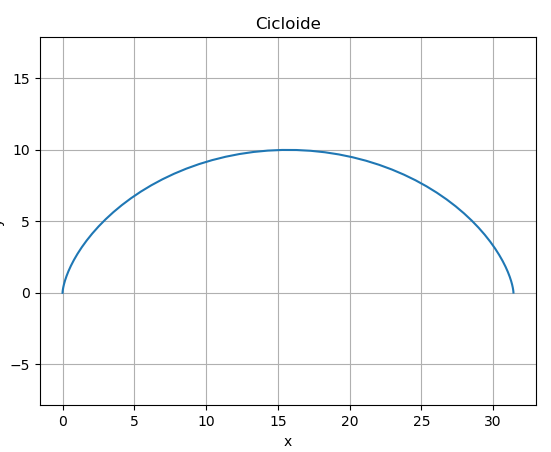
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
  
R = 5  
theta = np.arange(0, 2 \* np.pi, 0.01)  
  
x = R \* np.cos(theta)  
y = R \* np.sin(theta)  
plt.plot(x, y)  
  
plt.xlabel(**"x"**)  
plt.ylabel(**"y"**)  
plt.title(**"Circulo"**)  
plt.grid()  
plt.axis(**"equal"**)  
plt.savefig(**"plot\_polar\_circulo\_example.png"**)  
  
plt.show()



Os gráficos de barra servem principalmente para visualizar valores agrupados

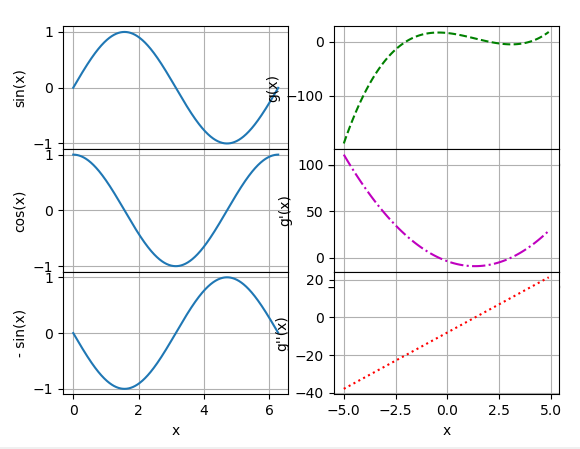
Exemplo Gráfico Ciclóide

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
  
LI = 0  
LS = 2 \* np.pi  
passo = 0.1  
R = 5  
  
*#Alterar com os alunos o passo  
# c = Origem O do circulo que desliza*c = np.arange(LI, LS, passo)  
  
x = R \* (c - np.sin(c))  
y = R \* (1 - np.cos(c))  
plt.plot(x, y)  
  
plt.xlabel(**"x"**)  
plt.ylabel(**"y"**)  
plt.title(**"Cicloide"**)  
plt.grid()  
plt.axis(**"equal"**)  
plt.savefig(**"plot\_polar\_cicloide\_example.png"**)  
  
plt.show()



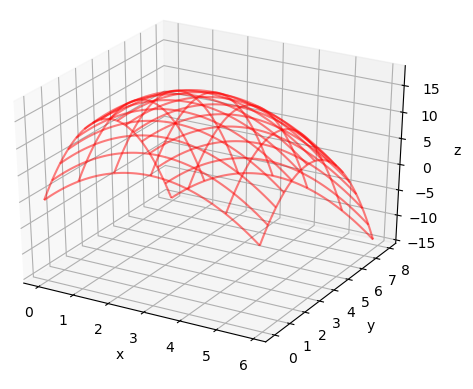
Exemplo Sub-Gráficos

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
  
plt.close(**"all"**)  
  
x = np.arange(0, 2 \* np.pi, 0.01)  
  
y = np.sin(x)  
y\_linha = np.cos(x)  
y\_duas\_linhas = - np.sin(x)  
  
x\_dois = np.arange(-5, 5, 0.1)  
  
g = x\_dois \*\* 3 - 4 \* x\_dois \*\* 2 - 4 \* x\_dois + 16  
g\_linha = 3 \* x\_dois \*\* 2 - 8 \* x\_dois - 4  
g\_linha\_dois = 6 \* x\_dois - 8  
  
*# figura = Figura que armazena o arranjo com os subgráficos  
# sharex = True (compartilhar o eixo x )  
# sharey = True (compartilhar o eixo y )*[figura, subgraficos] = plt.subplots(nrows=3, ncols=2, sharex=**False**, sharey=**False**)  
  
subgraficos[0, 0].plot(x, y)  
subgraficos[0, 0].set\_ylabel(**"sin(x)"**)  
subgraficos[0, 0].grid()  
  
subgraficos[1, 0].plot(x, y\_linha)  
subgraficos[1, 0].set\_ylabel(**"cos(x)"**)  
subgraficos[1, 0].grid()  
  
subgraficos[2, 0].plot(x, y\_duas\_linhas)  
subgraficos[2, 0].set\_xlabel(**"x"**)  
subgraficos[2, 0].set\_ylabel(**"- sin(x)"**)  
subgraficos[2, 0].grid()  
  
subgraficos[0, 1].plot(x\_dois, g,  
 color=**'g'**,  
 linestyle=**"--"**)  
subgraficos[0, 1].set\_ylabel(**"g(x)"**)  
subgraficos[0, 1].grid()  
  
subgraficos[1, 1].plot(x\_dois, g\_linha,  
 color=**'m'**,  
 linestyle=**"-."**)  
subgraficos[1, 1].set\_ylabel(**"g'(x)"**)  
subgraficos[1, 1].grid()  
  
subgraficos[2, 1].plot(x\_dois, g\_linha\_dois,  
 color=**'r'**,  
 linestyle=**":"**)  
subgraficos[2, 1].set\_xlabel(**"x"**)  
subgraficos[2, 1].set\_ylabel(**"g''(x)"**)  
subgraficos[2, 1].grid()  
  
figura.subplots\_adjust(hspace=0)  
plt.savefig(**"plot\_subgraficos\_sen\_cos.png"**)  
plt.show()



Exemplo Superfície 3D

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**from** mpl\_toolkits.mplot3d.axes3d **import** Axes3D  
  
x = np.arange(0, 6, 0.1)  
y = np.arange(0, 8, 0.1)  
  
*# np.meshgrid = Malha*[X, Y] = np.meshgrid(x, y)  
Z = - X \*\* 2 - Y \*\* 2 + 6 \* X + 6 \* Y  
  
plt.close(**"all"**)  
*# plt.figure(i) = Figura (Gráfico) com índice i*figura = plt.figure(1)  
*# add\_subplot = Tranformar o gráfico  
# padrão 2-D em uma projeção 3-D*axes = figura.add\_subplot(1, 1, 1, projection=**"3d"**)  
  
*# rstride = Distância na coordenada x  
# cstride = Distância na coordenada y  
# alpha = Espessura da linha  
# wireframe = Estrutura em arame*axes.plot\_wireframe(X, Y, Z,  
 color=**"r"**,  
 rstride=5,  
 cstride=10,  
 alpha=0.5)  
  
axes.set\_xlabel(**"x"**)  
axes.set\_ylabel(**"y"**)  
axes.set\_zlabel(**"z"**)  
  
plt.savefig(**"plot\_ponto\_maximo.png"**)  
  
plt.show()



TKINTER GUI

Uma biblioteca gráfica é um kit de ferramentas de software com uma coleção de classes que define uma funcionalidade de vários elementos da GUI. Essas bibliotecas de gráficos geralmente são escritas em C / C ++. Muitos deles foram portados para o Python na forma de módulos importáveis. Alguns deles estão listados abaixo:

|  |  |
| --- | --- |
| **Nome** | **Módulo** |
| Tkinter | É o kit de ferramentas Tcl-Tk GUI desenvolvido por Fredrik Lundh. Este módulo é fornecido com distribuições padrão do Python para todas as plataformas. |
| PyQt | O PyQt reúne a estrutura de aplicativos de plataforma cruzada Qt C ++ e a linguagem interpretada de plataforma cruzada Python.  Qt é mais do que um kit de ferramentas da GUI. Ele inclui abstrações de soquetes de rede, threads, Unicode, expressões regulares, bancos de dados SQL, SVG, OpenGL, XML, um navegador da Web totalmente funcional, um sistema de ajuda, uma estrutura multimídia e uma rica coleção de widgets da GUI. |
| [PyGTK](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt&u=https://pypi.org/project/PyGTK/&xid=17259,15700023,15700186,15700191,15700259,15700271,15700302&usg=ALkJrhhuw7cYr9kYtIDbtGrpS2Ogo9nh7w) | é outro popular módulo de GUI chamado GTK |
| WxWidgets | WxWidgets (<https://www.wxwidgets.org>) é uma biblioteca de classe GUI livre e portátil escrita em C ++ que fornece uma aparência e sensação nativas em várias plataformas |
| Kivy | *Kivy* é uma biblioteca GUI multi-plataforma que suporta sistemas operacionais de desktop (Windows, MacOS, Linux) e dispositivos móveis (Android, iOS). Está escrito em Python e Cython, e pode usar uma gama de backends de janelas. |
| PySide | O PySide é um wrapper do Python ao Qt |

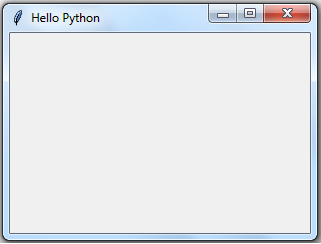
### Programação Tkinter

Tkinter é a biblioteca GUI padrão para Python. O Python, quando combinado com o Tkinter, fornece uma maneira rápida e fácil de criar aplicativos GUI. O Tkinter fornece uma poderosa interface orientada a objetos para o kit de ferramentas Tk GUI.

Criar um aplicativo GUI usando o Tkinter é uma tarefa fácil. Tudo o que você precisa fazer é executar as seguintes etapas -

* Importe o módulo Tkinter .
* Crie a janela principal do aplicativo GUI.
* Adicione um ou mais dos widgets mencionados acima ao aplicativo GUI.
* Digite o loop do evento principal para executar uma ação contra cada evento acionado pelo usuário.

Os elementos da GUI e suas funcionalidades são definidos no módulo Tkinter. O código a seguir demonstra as etapas na criação de uma interface do usuário. O código abaixo criará a seguinte janela:

[](https://www.tutorialsteacher.com/Content/images/python/window.png)

**from** tkinter **import** \*  
  
window = Tk()  
*# add widgets here*window.title(**'Hello Python'**)  
window.geometry(**"300x200+10+20"**)  
window.mainloop()

Antes de tudo, importe o módulo TKinter. Após a importação, configure o objeto do aplicativo chamando a função Tk() . Isso criará uma janela de nível superior (raiz) com um quadro com uma barra de título, uma caixa de controle com os botões minimizar e fechar e uma área do cliente para armazenar outros widgets.

O método geometry() define a largura, altura e coordenadas do canto superior esquerdo do quadro como abaixo (todos os valores estão em pixels): window.geometry("widthxheight+XPOS+YPOS")

O objeto do aplicativo entra em um evento escutando loop chamando o método mainloop() . O aplicativo agora está esperando constantemente por qualquer evento gerado nos elementos nele. O evento pode ser um texto digitado em um campo de texto, uma seleção feita no menu suspenso ou no botão de opção, ações de clique único/duplo do mouse etc.

A funcionalidade do aplicativo envolve a execução de funções de retorno de chamada apropriadas em resposta a um tipo específico de evento. Discutiremos a manipulação de eventos posteriormente neste tutorial. O loop de eventos terminará como e quando o botão Fechar na barra de título for clicado.

Todas as classes de widget Tkinter são herdadas da classe Widget. Vamos adicionar os widgets mais usados.

### Widgets Tkinter

O Tkinter fornece vários controles, como botões, rótulos e caixas de texto usadas em um aplicativo GUI. Esses controles são geralmente chamados de widgets.

Atualmente, existem 15 tipos de widgets no Tkinter. Apresentamos esses widgets, além de uma breve descrição na tabela a seguir

|  |  |
| --- | --- |
| **Operador** | **Operador e descrição** |
| [Button](https://www.tutorialspoint.com/python/tk_button.htm) | é usado para exibir botões no seu aplicativo. |
| [Canvas](https://www.tutorialspoint.com/python/tk_canvas.htm) | é usado para desenhar formas, como linhas, ovais, polígonos e retângulos, em seu aplicativo. |
| [Checkbutton](https://www.tutorialspoint.com/python/tk_checkbutton.htm) | é usado para exibir várias opções como caixas de seleção. O usuário pode selecionar várias opções ao mesmo tempo. |
| [Entry](https://www.tutorialspoint.com/python/tk_entry.htm) | é usado para exibir um campo de texto de linha única para aceitar valores de um usuário. |
| [Frame](https://www.tutorialspoint.com/python/tk_frame.htm) | é usado como um widget de contêiner para organizar outros widgets. |
| [Label](https://www.tutorialspoint.com/python/tk_label.htm) | é usado para fornecer uma legenda de linha única para outros widgets. Também pode conter imagens. |
| [Listbox](https://www.tutorialspoint.com/python/tk_listbox.htm) | é usado para fornecer uma lista de opções para um usuário. |
| [Menubutton](https://www.tutorialspoint.com/python/tk_menubutton.htm) | é usado para exibir menus no seu aplicativo. |
| [Menu](https://www.tutorialspoint.com/python/tk_menu.htm) | é usado para fornecer vários comandos para um usuário. Esses comandos estão contidos no Menubutton. |
| [Message](https://www.tutorialspoint.com/python/tk_message.htm) | é usado para exibir campos de texto com várias linhas para aceitar valores de um usuário. |
| [Radiobutton](https://www.tutorialspoint.com/python/tk_radiobutton.htm) | é usado para exibir várias opções como botões de opção. O usuário pode selecionar apenas uma opção por vez. |
| [Scale](https://www.tutorialspoint.com/python/tk_scale.htm) | é usado para fornecer um widget deslizante. |
| [Scrollbar](https://www.tutorialspoint.com/python/tk_scrollbar.htm) | é usado para adicionar capacidade de rolagem a vários widgets, como caixas de listagem. |
| [Text](https://www.tutorialspoint.com/python/tk_text.htm) | é usado para exibir texto em várias linhas. |
| [Toplevel](https://www.tutorialspoint.com/python/tk_toplevel.htm) | é usado para fornecer um contêiner de janela separado. |
| [Spinbox](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt&u=https://www.tutorialspoint.com/python/tk_spinbox.htm&usg=ALkJrhhIrgkmw-9KA3tjePF-DKlzjRHd8Q) | é uma variante do Tkinter entrada padrão widget, que pode ser utilizado para seleccionar a partir de um número fixo de valores. |
| PanedWindow | é um widget de contêiner que pode conter qualquer número de painéis, organizados horizontal ou verticalmente. |
| [LabelFrame](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt&u=https://www.tutorialspoint.com/python/tk_labelframe.htm&usg=ALkJrhiB4ydvPFy09YoXKVHtcJg7M-dfGA) | é um widget de contêiner simples. Seu principal objetivo é atuar como espaçador ou contêiner para layouts de janelas complexos. |
| [tkMessageBox](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt&u=https://www.tutorialspoint.com/python/tk_messagebox.htm&usg=ALkJrhjAkMysYc4e99va3LW_5B1HFbDy6A) | é usado para exibir caixas de mensagens em seus aplicativos. |

Atributos padrão dar uma olhada em como alguns de seus atributos comuns são especificados, como tamanhos, cores e fontes.

Dimensões

Vários comprimentos, larguras e outras dimensões de widgets podem ser descritos em muitas unidades diferentes. Se você definir uma dimensão como um número inteiro, será assumido que está em pixels.

Você pode especificar unidades definindo uma dimensão para uma sequência de um número seguido por.

|  |  |
| --- | --- |
| **Caracter** | **Descrição** |
| c | Centimeters |
| i | Inches |
| m | Millimeters |
| p | Printer's points (about 1/72") |

Opções de comprimento

Tkinter expressa um comprimento como um número inteiro de pixels. Aqui está a lista de opções comuns de comprimento:

|  |  |
| --- | --- |
| **Propriedade** | **Descrição** |
| borderwidth | Largura da borda que dá uma aparência tridimensional ao widget. |
| *highlightthickness* | Largura do retângulo do realce quando o widget está focado. |
| padX padY | Espaço extra que o widget solicita ao seu gerenciador de layout além do mínimo que o widget precisa para exibir seu conteúdo nas direções xe y. |
| selectborderwidth | Largura da borda tridimensional em torno dos itens selecionados do widget. |
| wraplength | Comprimento máximo da linha para widgets que executam quebra de linha. |
| height | Altura desejada do widget; deve ser maior ou igual a 1. |
| underline | Índice do caractere a ser sublinhado no texto do widget (0 é o primeiro caractere, 1 o segundo e assim por diante). |
| width | largura desejada do widget. |

Cores

Tkinter representa cores com strings. Existem duas formas básicas de representar cores no Tkinter. Como uma string hexadecimal e através do nome da cor:

|  |  |
| --- | --- |
| **Hexadecimal** | **Nome da cor** |
| #FFFFFF | white |
| #000000 | black |
| #FF0000 | red |
| #00FF00 | green |
| #0000FF | blue |
| #00FFFF | cyan |
| #FFFF00 | yellow |
| #FF00FF | magenta |

As opções de cores comuns são;

|  |  |
| --- | --- |
| **Propriedade** | **Descrição** |
| activebackground | Cor de fundo do widget quando o widget está ativo. |
| activeforeground | Cor de primeiro plano do widget quando o widget está ativo. |
| background | Cor de fundo do widget. Isso também pode ser representado como bg . |
| disabledforeground | Cor de primeiro plano para o widget quando o widget está desativado. |
| foreground | Cor do primeiro plano do widget. Pode também ser representado como fg . |
| highlightsbackground | Cor de fundo da região de destaque quando o widget está focado. |
| highlightscolor | Cor de primeiro plano da região de destaque quando o widget estiver focado. |
| selectbackground | Cor do plano de fundo para os itens selecionados do widget. |
| selectforeground | Cor de primeiro plano para os itens selecionados do widget. |

Fontes

Dependendo do sistema operacional, existem três diferentes maneiras de se especificar um font no Tkinter.

### 1- Através de tuplas

Você especifica o font como uma tupla cujo primeiro elemento é a família do font, seguido to tamanho em points, e opcionalmente um modificador escolhido entre os seguintes estilos disponíveis: bold (negrito) , italic (itálico) , underline (sublinhado) e overstrike (tachado).

* ("Helvetica", "16") para um Helvetica regular de 16 pontos.
* ("Times", "24", "bold itálico") para um Times de 24 pontos em itálico.

### 2 - Através de objetos “Fonts”

Para criar um objeto font você importa o módulo tkFonte e utiliza o construtor.

import tkFont

font = tkFont.Font ( option, ... )

Exemplo:

helv24 = tkFont.Font(family=”Helvetica”,size=246,weight=”bold”,undeline= 1)

A lista de opções para o construtor é a seguinte:

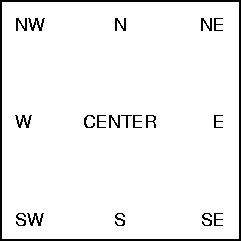
|  |  |
| --- | --- |
| **Propriedade** | **Descrição** |
| family | O nome da família da fonte como uma string |
| size | A altura da fonte como um número inteiro em pontos. Para obter uma fonte com n pixels de altura, use -n. |
| weight | “bold" para negrito, "normal" para peso regular. |
| slant | "italic " para itálico, "roman" para normal. |
| underline | 1 para texto sublinhado, 0 para normal. |
| overstrike | 1 para texto com overtruck, 0 para normal. |

3 - Fontes X Window

Você pode usar qualquer nome de font disponível no seu sistema. Para saber quais são as fontes disponíveis utilize o programa sfontsel. Mas observe que uma fonte específica pode não estar disponível em todos os sistemas. Se a fonte não estiver disponível, o Tkinter irá procurar encontrar a mais próxima, e se tiver nada parecido, vai utilizar o font “default” do sistema.

Anchors

As âncoras são usadas para definir onde o texto é posicionado em relação a um ponto de referência. As constantes da âncora são mostradas neste diagrama -



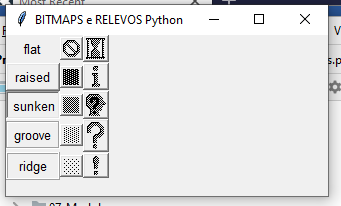
ScrollBar

**from** tkinter **import** \*  
  
top = Tk()  
sb = Scrollbar(top)  
sb.pack(side=RIGHT, fill=Y)  
  
mylist = Listbox(top, yscrollcommand=sb.set)  
  
**for** line **in** range(30):  
 mylist.insert(END, **"Number "** + str(line))  
  
mylist.pack(side=LEFT)  
sb.config(command=mylist.yview)  
  
mainloop()

Estilos de Relevo e Bitmaps

O estilo de relevo de um widget refere-se a certos efeitos 3D simulados na parte externa do widget. Já o Bitmap adiciona uma imagem.

**from** tkinter **import** \*  
  
Relevos = [FLAT, RAISED, SUNKEN, GROOVE, RIDGE]  
  
Bitmaps = [**'error'**, **'gray75'**, **'gray50'**, **'gray25'**, **'gray12'**,  
 **'hourglass'**, **'info'**, **'questhead'**, **'question'**, **'warning'**]  
  
window = Tk()  
  
window.title(**'BITMAPS e RELEVOS Python'**)  
  
row = 0  
col = 0  
**for** bit **in** Relevos:  
 e = Button(window, text=bit, relief=bit, font=(**None**, 10))  
 e.grid(row=row, column=col, sticky=E + W)  
 row += 1  
  
**for** rel **in** Bitmaps:  
 **if** (row > 4):  
 row = 0  
 col += 1  
  
 e = Button(window, text=rel, bitmap=rel, relief=RAISED, font=(**None**, 10))  
 e.grid(row=row, column=col, sticky=E + W)  
 row += 1  
  
window.geometry(**"300x200+10+20"**)  
window.mainloop()



Cursores

O Python Tkinter suporta vários cursores de mouse diferentes disponíveis. O gráfico exato pode variar de acordo com o seu sistema operacional. Tente o exemplo a seguir movendo o cursor em botões diferentes.

**from** tkinter **import** \*  
  
CURSORES = [**'arrow'**, **'exchange'**, **'pirate'**, **'spraycan'**, **'watch'**, **'circle'**,  
 **'fleur'**, **'plus'**, **'star'**, **'clock'**, **'heart'**, **'shuttle'**,  
 **'target'**, **'cross'**, **'man'**, **'sizing'**, **'tcross'**, **'dotbox'**,  
 **'mouse'**, **'spider'**, **'trek'**]  
  
window = Tk()  
  
window.title(**'CURSORES Python'**)

row = 0  
col = 0  
**for** cursor **in** CURSORES:  
 e = Button(window, text=cursor, cursor=cursor,  
 font=(**None**, 10))  
 e.grid(row=row, column=col, sticky=E + W)  
 row += 1  
 **if** (row > 4):  
 row = 0  
 col += 1  
  
window.geometry(**"300x200+10+20"**)  
window.mainloop()

Gerenciamento de geometria

Todos os widgets do Tkinter têm acesso a métodos específicos de gerenciamento de geometria, que têm o objetivo de organizar widgets em toda a área do widget pai. O Tkinter expõe as seguintes classes do gerenciador de geometria: pack, grid e place.

* pack() - organiza os widgets em blocos antes de colocá-los no widget pai.
* grid() - organiza widgets em uma estrutura semelhante a uma tabela no widget pai.
* place() - organiza os widgets colocando-os em uma posição específica no widget pai.

### Botão

O botão pode ser criado usando a classe Button. O construtor da classe Button requer uma referência à janela principal e às opções.

Assinatura: Button(window, attributes)

Você pode definir as seguintes propriedades importantes para personalizar um botão:

|  |  |
| --- | --- |
| **Propriedade** | **Descrição** |
| text | legenda do botão |
| bg | cor de fundo |
| fg | cor de primeiro plano |
| font | nome e tamanho da fonte |
| image | a ser exibido em vez de texto |
| command | função a ser chamada quando clicada |
| activebackground | Cor de fundo quando o botão está sob o cursor. |
| activeforeground | Cor de primeiro plano quando o botão está sob o cursor. |
| borderwidth | Largura da borda em pixels. O padrão é 2. |
| bd | Cor de fundo normal. |
| command | Função ou método a ser chamado quando o botão é clicado. |
| fg | Cor normal de primeiro plano (texto). |
| font | Fonte de texto a ser usada para o rótulo do botão. |
| height | Altura do botão em linhas de texto (para botões de texto) ou pixels (para imagens). |
| highlightcolor | A cor do foco é realçada quando o widget está em foco. |
| image | Imagem a ser exibida no botão (em vez de texto). |
| justify | Como mostrar várias linhas de texto: ESQUERDA para justificar à esquerda cada linha; CENTRO para centralizá-los; ou DIREITO para justificar à direita. |
| padx | Preenchimento adicional à esquerda e à direita do texto. |
| pady | Preenchimento adicional acima e abaixo do texto. |
| relief | Alívio especifica o tipo da borda. Alguns dos valores são SUNKEN, RAISED, GROOVE e RIDGE. |
| state | Defina esta opção como DESATIVADO para acinzentar o botão e deixar de responder. Tem o valor ATIVO quando o mouse está sobre ele. O padrão é NORMAL. |
| underline | O padrão é -1, o que significa que nenhum caractere do texto no botão será sublinhado. Se não negativo, o caractere de texto correspondente será sublinhado. |
| width | Largura do botão em letras (se estiver exibindo texto) ou pixels (se estiver exibindo uma imagem). |
| wraplength | Se esse valor for definido como um número positivo, as linhas de texto serão quebradas para caber nesse comprimento. |

A seguir, são métodos comumente usados ​​para este widget –

|  |  |
| --- | --- |
| **Método** | **Descrição** |
| flash() | Faz com que o botão pisque várias vezes entre as cores ativa e normal. Deixa o botão no estado em que estava originalmente. Ignorado se o botão estiver desativado. |
| invoke() | Chama o retorno de chamada do botão e retorna o que essa função retorna. Não tem efeito se o botão estiver desativado ou se não houver retorno de chamada. |

**from** tkinter **import** \*  
  
window = Tk()  
btn=Button(window, text=**"This is Button widget"**, fg=**'blue'**)  
btn.place(x=80, y=100)  
  
window.title(**'Hello Python'**)  
window.geometry(**"300x200+10+20"**)  
window.mainloop()

### Label

Um label pode ser criado na interface do usuário em Python usando a classe Label. O construtor Label requer o objeto da janela de nível superior e os parâmetros de opções. Os parâmetros da opção são semelhantes ao objeto Button. A seguir, um label é adicionado na janela.

|  |  |
| --- | --- |
| **Propriedade** | **Descrição** |
| anchor | Esta opção controla onde o texto está posicionado se o widget tiver mais espaço do que o texto precisa. O padrão é anchor = CENTER, que centraliza o texto no espaço disponível. |
| bg | A cor de fundo normal exibida atrás do rótulo e do indicador. |
| bitmap | Defina esta opção igual a um bitmap ou objeto de imagem e o rótulo exibirá esse gráfico. |
| bd | O tamanho da borda ao redor do indicador. O padrão é 2 pixels. |
| cursor | Se você definir esta opção como um nome de cursor ( *seta, ponto etc.* ), o cursor do mouse mudará para esse padrão quando estiver sobre o botão de verificação. |
| font | Se você estiver exibindo texto nesse rótulo (com a opção texto ou variável de texto, a opção fonte especifica em qual fonte esse texto será exibido. |
| fg | Se você estiver exibindo texto ou um bitmap neste rótulo, esta opção especificará a cor do texto. Se você estiver exibindo um bitmap, essa é a cor que aparecerá na posição dos 1 bits no bitmap. |
| altura | A dimensão vertical do novo quadro. |
| image | Para exibir uma imagem estática no widget de etiqueta, defina esta opção para um objeto de imagem. |
| justify | Especifica como várias linhas de texto serão alinhadas uma à outra: ESQUERDA para nivelar à esquerda, CENTER para centralizar (o padrão) ou DIREITA para justificar à direita. |
| padx | Espaço extra adicionado à esquerda e à direita do texto no widget. O padrão é 1. |
| pady | Espaço extra adicionado acima e abaixo do texto no widget. O padrão é 1. |
| relief | Especifica a aparência de uma borda decorativa ao redor do rótulo. O padrão é FLAT; para outros valores. |
| text | Para exibir uma ou mais linhas de texto em um widget de etiqueta, configure esta opção para uma sequência que contenha o texto. Novas linhas internas ("\ n") forçarão uma quebra de linha. |
| textvariable | Para escravizar o texto exibido em um widget de etiqueta em uma variável de controle da classe *StringVar* , configure esta opção para essa variável. |
| underline | Você pode exibir um sublinhado (\_) abaixo da enésima letra do texto, contando de 0, configurando esta opção para n. O padrão é sublinhado = -1, o que significa que não há sublinhado. |
| width | Largura do rótulo em caracteres (não pixels!). Se essa opção não estiver definida, o rótulo será dimensionado para caber em seu conteúdo. |
| wraplength | Você pode limitar o número de caracteres em cada linha, configurando esta opção para o número desejado. O valor padrão, 0, significa que as linhas serão quebradas apenas nas novas linhas. |

Abaixo, a legenda do label será exibida em vermelho usando a fonte Helvetica de 16 pontos.

**from** tkinter **import** \*  
  
window = Tk()  
lbl = Label(window,  
 text=**"This is Label widget"**,  
 fg=**'red'**,  
 font=(**"Helvetica"**, 16))  
lbl.place(x=60, y=50)  
window.title(**'Hello Python'**)  
window.geometry(**"300x200+10+10"**)  
window.mainloop()

### Entry

Este widget renderiza uma caixa de texto de linha única para aceitar a entrada do usuário. Para entrada de texto com várias linhas, use o widget Texto. Além das propriedades já mencionadas, o construtor da classe Entry aceita o seguinte:

* bd: tamanho da borda da caixa de texto; o padrão é 2 pixels.
* show: para converter a caixa de texto em um campo de senha, defina show property para "\*".

O código a seguir adiciona o campo de texto.

txtfld=Entry(window, text="This is Entry Widget", bg='black',fg='white', bd=5)

O exemplo a seguir cria uma janela com um botão, rótulo e campo de entrada.

[](https://www.tutorialsteacher.com/Content/images/python/ui-widgets.png)**from** tkinter **import** \*  
  
window = Tk()  
btn = Button(window, text=**"This is Button widget"**, fg=**'blue'**)  
btn.place(x=80, y=100)  
lbl = Label(window, text=**"This is Label widget"**, fg=**'red'**, font=(**"Helvetica"**, 16))  
lbl.place(x=60, y=50)  
txtfld = Entry(window, text=**"This is Entry Widget"**, bd=5)  
txtfld.place(x=80, y=150)  
window.title(**'Hello Python'**)  
window.geometry(**"300x200+10+10"**)  
window.mainloop()

|  |  |
| --- | --- |
| **Propriedade** | **Descrição** |
| bg | A cor de fundo normal exibida atrás do rótulo e do indicador. |
| bd | O tamanho da borda ao redor do indicador. O padrão é 2 pixels. |
| command | Um procedimento a ser chamado toda vez que o usuário altera o estado desse botão de verificação. |
| cursor | Se você definir esta opção como um nome de cursor ( *seta, ponto etc.* ), o cursor do mouse mudará para esse padrão quando estiver sobre o botão de verificação. |
| font | A fonte usada para o texto. |
| exportselection | Por padrão, se você selecionar texto em um widget de Entrada, ele será exportado automaticamente para a área de transferência. Para evitar essa exportação, use exportelection = 0. |
| fg | A cor usada para renderizar o texto. |
| highlightcolor | A cor do foco é realçada quando o botão de verificação está com o foco. |
| justify | Se o texto contiver várias linhas, essa opção controlará como o texto é justificado: CENTRO, ESQUERDA ou DIREITA. |
| relief | Com o valor padrão, relief = FLAT, o botão de verificação não se destaca do fundo. Você pode definir esta opção para qualquer um dos outros estilos |
| selectbackground | A cor de fundo a ser usada na exibição do texto selecionado. |
| selectborderwidth | A largura da borda a ser usada em torno do texto selecionado. O padrão é um pixel. |
| selectforeground | A cor de primeiro plano (texto) do texto selecionado. |
| show | Normalmente, os caracteres que o usuário digita aparecem na entrada. Para criar uma senha. entrada que ecoa cada caractere como um asterisco, defina show = "\*". |
| state | O padrão é state = NORMAL, mas você pode usar state = DISABLED para desativar o controle e torná-lo sem resposta. Se o cursor estiver sobre o botão de verificação, o estado é ATIVO. |
| textvariable | Para poder recuperar o texto atual do seu widget de entrada, você deve configurar esta opção para uma instância da classe StringVar. |
| width | A largura padrão de um botão de verificação é determinada pelo tamanho da imagem ou texto exibido. Você pode definir esta opção para um número de caracteres e o botão de seleção sempre terá espaço para tantos caracteres. |
| xscrollcommand | Se você espera que os usuários frequentemente digitem mais texto que o tamanho da tela do widget, é possível vincular seu widget de entrada a uma barra de rolagem. |

A seguir, são métodos comumente usados ​​para este widget -

|  |  |
| --- | --- |
| **Método** | **Descrição** |
| delete ( first, last=None ) | Exclui caracteres do widget, começando pelo primeiro no índice, até mas sem incluir o último na posição. Se o segundo argumento for omitido, somente o caractere único na posição primeiro será excluído. |
| get() | Retorna o texto atual da entrada como uma sequência. |
| icursor ( index ) | Defina o cursor de inserção imediatamente antes do caractere no índice especificado. |
| index ( index ) | Mude o conteúdo da entrada para que o caractere no índice especificado seja o caractere visível mais à esquerda. Não tem efeito se o texto couber inteiramente na entrada. |
| insert ( index, s ) | Insere a sequência s antes do caractere no índice especificado. |
| select\_adjust ( index ) | Este método é usado para garantir que a seleção inclua o caractere no índice especificado. |
| select\_clear() | Limpa a seleção. Se não houver atualmente uma seleção, não terá efeito. |
| select\_from ( index ) | Define a posição do índice ANCHOR para o caractere selecionado pelo índice e seleciona esse caractere. |
| select\_present() | Se houver uma seleção, retorna true, caso contrário, retorna false. |
| select\_range ( start, end ) | Define a seleção sob controle do programa. Seleciona o texto começando no índice inicial, até mas sem incluir o caractere no índice final. A posição inicial deve estar antes da posição final. |
| select\_to ( index ) | Seleciona todo o texto da posição ANCHOR até, mas não incluindo, o caractere no índice especificado. |
| xview ( index ) | Este método é útil para vincular o widget Entrada a uma barra de rolagem horizontal. |
| xview\_scroll ( number, what ) | Usado para rolar a entrada horizontalmente. O argumento what deve ser UNITS, para rolar por larguras de caracteres, ou PAGES, para rolar em pedaços o tamanho do widget de entrada. O número é positivo para rolar da esquerda para a direita, negativo para rolar da direita para a esquerda. |

### Widgets de seleção

Radiobutton:

Este widget exibe um botão de alternância com o estado ON/OFF. Pode haver mais de um botão, mas apenas um deles estará LIGADO em um determinado momento.

Checkbutton:

Também é um botão de alternância. Uma caixa de seleção retangular aparece antes da legenda. Seu estado LIGADO é exibido pela marca de seleção na caixa que desaparece quando se clica em DESLIGADO.

Combobox:

Esta classe é definida no módulo ttk do tkinterpackage. Ele preenche dados suspensos de um tipo de dados de coleção, como uma tupla ou uma lista como parâmetro de valores.

Listbox:

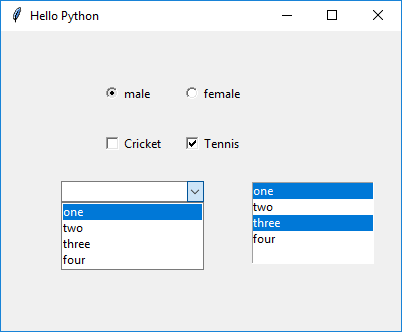
Ao contrário da Combobox, esse widget exibe toda a coleção de itens de sequência. O usuário pode selecionar um ou vários itens.

|  |  |
| --- | --- |
| **Propriedade** | **Descrição** |
| **bg** | A cor de fundo normal exibida atrás do rótulo e do indicador. |
| **bd** | O tamanho da borda ao redor do indicador. O padrão é 2 pixels. |
| **cursor** | O cursor que aparece quando o mouse está sobre a caixa de listagem. |
| **font** | A fonte usada para o texto na caixa de listagem. |
| **fg** | A cor usada para o texto na caixa de listagem. |
| **height** | Número de linhas (não pixels!) Mostradas na caixa de listagem. O padrão é 10. |
| **highlightcolor** | As cores mostradas no foco destacam quando o widget tem o foco. |
| **highlightthickness** | Espessura do destaque do foco. |
| **relief** | Seleciona efeitos de sombreamento tridimensional de borda. O padrão é SUNKEN. |
| **selectbackground** | A cor de fundo a ser usada na exibição do texto selecionado. |
| **selectmode** | Determina quantos itens podem ser selecionados e como os arrastamentos do mouse afetam a seleção –  BROWSE - Normalmente, você pode selecionar apenas uma linha de uma caixa de listagem. Se você clicar em um item e arrastar para uma linha diferente, a seleção seguirá o mouse. Esse é o padrão.  SINGLE - Você pode selecionar apenas uma linha e não pode arrastar o mouse. Sempre que clicar no botão 1, essa linha é selecionada.  MULTIPLE - Você pode selecionar qualquer número de linhas ao mesmo tempo. Clicar em qualquer linha alterna se está ou não selecionado.  EXTENDED - Você pode selecionar qualquer grupo adjacente de linhas de uma só vez, clicando na primeira linha e arrastando para a última linha. |
| **width** | A largura do widget em caracteres. O padrão é 20. |
| **xscrollcommand** | Se você quiser permitir que o usuário role a caixa de listagem horizontalmente, poderá vincular o widget da caixa de listagem a uma barra de rolagem horizontal. |
| **yscrollcommand** | Se você quiser permitir que o usuário role a caixa de listagem verticalmente, poderá vincular o widget da caixa de listagem a uma barra de rolagem vertical. |

Métodos em objetos de caixa de listagem incluem –

|  |  |
| --- | --- |
| **Método** | **Descrição** |
| **activate ( index )** | Seleciona a linha especificada pelo índice especificado. |
| **curselection()** | Retorna uma tupla contendo os números de linha do (s) elemento (s) selecionado (s), contando de 0. Se nada for selecionado, retornará uma tupla vazia. |
| **delete ( first, last=None )** | Exclui as linhas cujos índices estão no intervalo [primeiro, último]. Se o segundo argumento for omitido, a linha única com o índice primeiro será excluída. |
| **get ( first, last=None )** | Retorna uma tupla contendo o texto das linhas com índices do primeiro ao último, inclusive. Se o segundo argumento for omitido, retornará o texto da linha mais próxima da primeira. |
| **index ( i )** | Se possível, posiciona a parte visível da caixa de listagem para que a linha que contém o índice i esteja na parte superior do widget. |
| **insert ( index, \*elements )** | Insira uma ou mais novas linhas na caixa de listagem antes da linha especificada pelo índice. Use END como o primeiro argumento se desejar adicionar novas linhas ao final da caixa de listagem. |
| **nearest ( y )** | Retorne o índice da linha visível mais próxima da coordenada y relativa ao widget da caixa de listagem. |
| **see ( index )** | Ajuste a posição da caixa de listagem para que a linha mencionada pelo índice fique visível. |
| **size()** | Retorna o número de linhas na caixa de listagem. |
| **xview()** | Para tornar a caixa de listagem horizontalmente rolável, defina a opção de comando da barra de rolagem horizontal associada a este método. |
| **xview\_moveto ( fraction )** | Role a caixa de listagem para que a fração mais à esquerda da largura de sua linha mais longa fique fora do lado esquerdo da caixa de listagem. A fração está no intervalo [0,1]. |
| **xview\_scroll ( number, what )** | Rola a caixa de listagem horizontalmente. Para o argumento what, use UNITS para rolar por caracteres ou PAGES para rolar por páginas, ou seja, pela largura da caixa de listagem. O argumento numérico informa quantos rolagem. |
| **yview()** | Para tornar a caixa de listagem rolável verticalmente, defina a opção de comando da barra de rolagem vertical associada a esse método. |
| **yview\_moveto ( fraction )** | Role a caixa de listagem para que a fração superior da largura de sua linha mais longa fique fora do lado esquerdo da caixa de listagem. A fração está no intervalo [0,1]. |

O exemplo a seguir demonstra a janela com os widgets de seleção: Radiobutton, Checkbutton, Listbox e Combobox:

[](https://www.tutorialsteacher.com/Content/images/python/window2.png)

**from** tkinter **import** \*  
**from** tkinter.ttk **import** Combobox  
  
window = Tk()  
var = StringVar()  
var.set(**"one"**)  
data = (**"one"**, **"two"**, **"three"**, **"four"**)  
cb = Combobox(window, values=data)  
cb.place(x=60, y=150)  
  
lb = Listbox(window, height=5, selectmode=**'multiple'**)  
**for** num **in** data:  
 lb.insert(END, num)  
lb.place(x=250, y=150)  
  
v0 = IntVar()  
v0.set(2)  
r1 = Radiobutton(window, text=**"male"**, variable=v0, value=1)  
r2 = Radiobutton(window, text=**"female"**, variable=v0, value=2)  
r1.place(x=100, y=50)  
r2.place(x=180, y=50)  
  
v1 = IntVar()  
v2 = IntVar()  
C1 = Checkbutton(window, text=**"Cricket"**, variable=v1)  
C2 = Checkbutton(window, text=**"Tennis"**, variable=v2)  
C1.place(x=100, y=100)  
C2.place(x=180, y=100)  
  
window.resizable(width=**False**,height=**False**)  
window.title(**"Conversor de Temperatura."**)  
window.geometry(**"500x300"**)  
window.mainloop()

## Manipulação de eventos

Um evento é uma notificação recebida pelo objeto do aplicativo de vários widgets da GUI como resultado da interação do usuário. O objeto Application está sempre antecipando eventos, pois executa um loop de escuta de eventos. As ações do usuário incluem clique no botão do mouse ou clique duplo, tecla do teclado pressionada enquanto o controle está dentro da caixa de texto, certos elementos ganham ou ficam fora de foco etc.

Eventos são expressos como cadeias de caracteres no formato <modifier-type-qualifier>.

Muitos eventos são representados apenas como qualificador. O tipo define a classe do evento.

A tabela a seguir mostra como o Tkinter reconhece diferentes eventos:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Event** | **Modifier** | **Type** | **Qualifier** | **Action** |
| <Button-1> |  | Button | 1 | Clique com o botão esquerdo do mouse. |
| <Button-2> |  | Button | 2 | Clique no botão do meio do mouse. |
| <Destroy> |  | Destroy |  | Janela está sendo destruída. |
| <Double-Button-1> | Double | Button | 1 | Clique duas vezes o primeiro botão do mouse 1. |
| <Enter> | Enter |  |  | O cursor entra na janela. |
| <Expose> |  | Expose |  | Janela totalmente ou parcialmente exposta. |
| <KeyPress-a> |  | KeyPress | a | Qualquer tecla foi pressionada. |
| <KeyRelease> |  | KeyRelease |  | Qualquer tecla foi liberada. |
| <Leave> |  | Leave |  | O cursor sai da janela. |
| <Print> |  |  | Print | A tecla PRINT foi pressionada. |
| <FocusIn> |  | FocusIn |  | Controle ganha foco |
| <FocusOut> |  | FocusOut |  | Controle perde o foco. |

Um evento deve ser registrado com um ou mais widgets da GUI no aplicativo. Caso contrário, será ignorado. No Tkinter, existem duas maneiras de registrar um evento com um widget. A primeira maneira é usar o método bind() e a segunda maneira usando o parâmetro command no construtor de widget.

### Método Bind ()

O método bind() associa um evento a uma função de retorno de chamada para que, quando ocorrer o par, a função seja chamada.

Sintaxe:

Widget.bind (evento, retorno de chamada)

Por exemplo, para invocar a função MyButtonClicked() no botão esquerdo do mouse, use o seguinte código:

Exemplo: Even Binding

Example: Even Binding

from tkinter import \*

window=Tk()

btn = Button(window, text='OK')

btn.bind('<Button-1>', MyButtonClicked)

O objeto de evento é caracterizado por muitas propriedades, como widget de origem, coordenadas de posição, número do botão do mouse e tipo de evento. Eles podem ser passados ​​para a função de retorno de chamada, se necessário.

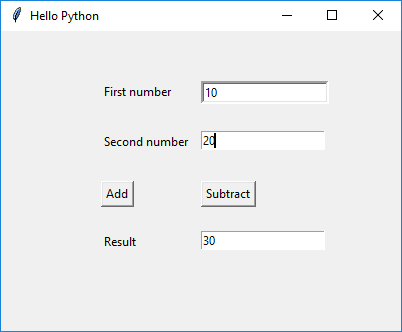
## Parâmetro de comando

Cada widget responde principalmente a um tipo específico. Por exemplo, Button é uma fonte do evento Button. Portanto, é por padrão vinculado a ele. Os métodos construtores de muitas classes de widgets possuem um parâmetro opcional chamado command. Este parâmetro de comando é definido para retornar a função que será chamada sempre que ocorrer um evento vinculado. Este método é mais conveniente que o método bind() .

btn = Button(window, text='OK', command=myEventHandlerFunction)

No exemplo abaixo, a janela do aplicativo possui dois campos de entrada de texto e outro para exibir o resultado. Existem dois objetos de botão com as legendas Adicionar e Subtrair. Espera-se que o usuário insira o número nos dois widgets de Entrada. Sua adição ou subtração é exibida no terceiro.

O primeiro botão (Adicionar) é configurado usando o parâmetro de comando. Seu valor é o método add() na classe. O segundo botão usa o método bind() para registrar o clique do botão esquerdo com o método sub() . Ambos os métodos leem o conteúdo dos campos de texto pelo método get() do widget Entry, analisam números, executam a adição / subtração e exibem o resultado no terceiro campo de texto usando o método insert() .

[](https://www.tutorialsteacher.com/Content/images/python/window3.png)**from** tkinter **import** \*  
  
**class** MyWindow:  
 **def** \_\_init\_\_(self, win):  
 self.lbl1 = Label(win, text=**'First number'**)  
 self.lbl2 = Label(win, text=**'Second number'**)  
 self.lbl3 = Label(win, text=**'Result'**)  
  
 self.lbl1.place(x=100, y=50)  
 self.lbl2.place(x=100, y=100)  
 self.lbl3.place(x=100, y=200)  
  
 self.t1 = Entry(bd=3)  
 self.t2 = Entry()  
 self.t3 = Entry()  
 *## self.btn1 = Button(win, text='Add')  
 ## self.btn2=Button(win, text='Subtract')* self.t1.place(x=200, y=50)  
 self.t2.place(x=200, y=100)  
 self.t3.place(x=200, y=200)  
  
 self.b1 = Button(win, text=**'Add'**, command=self.add)  
 self.b2 = Button(win, text=**'Subtract'**)  
 self.b2.bind(**'<Button-1>'**, self.sub)  
 self.b1.place(x=100, y=150)  
 self.b2.place(x=200, y=150)  
  
 **def** add(self):  
 self.t3.delete(0, **'end'**)  
 num1 = int(self.t1.get())  
 num2 = int(self.t2.get())  
 result = num1 + num2  
 self.t3.insert(END, str(result))  
  
 **def** sub(self, event):  
 self.t3.delete(0, **'end'**)  
 num1 = int(self.t1.get())  
 num2 = int(self.t2.get())  
 result = num1 - num2  
 self.t3.insert(END, str(result))  
  
window = Tk()  
mywin = MyWindow(window)  
window.title(**'Hello Python'**)  
window.geometry(**"400x300+10+10"**)  
window.mainloop()

O exemplo acima cria a seguinte interface do usuário.

Assim, você pode criar a interface do usuário usando o TKinter em Python.

Outras bibliotecas Python

Um bom site para procurar um modulo e o Python Package Index- PyPI (https://pypi.org).

Outro projeto que não pode deixar de ser visitado e o Pycairo (https://www.cairographics.org/pycairo/), que permite o desenho de gráficos com recursos avançados, como suavização de curvas e transparências.

Para trabalhar com imagens (PNG, JPG etc.), o modulo PILLOW, ou fork do Python Image Library (https://python-pillow.org), é referência.

Se você deseja adicionar cores a seu console, imprimindo mensagens com cores diferentes, limpando a tela e mesmo trabalhando com o teclado de forma mais direta, visite a ColorConsole (<https://github.com/lskbr/colorconsole>).