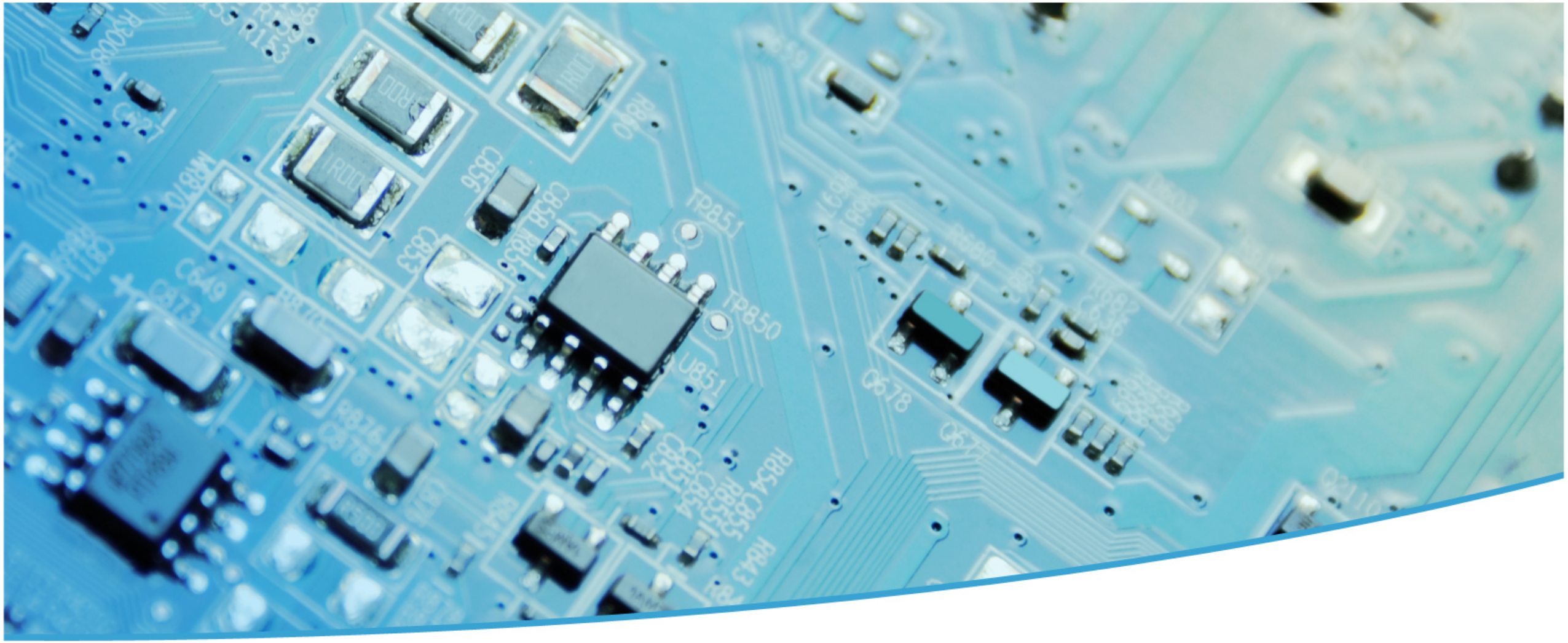
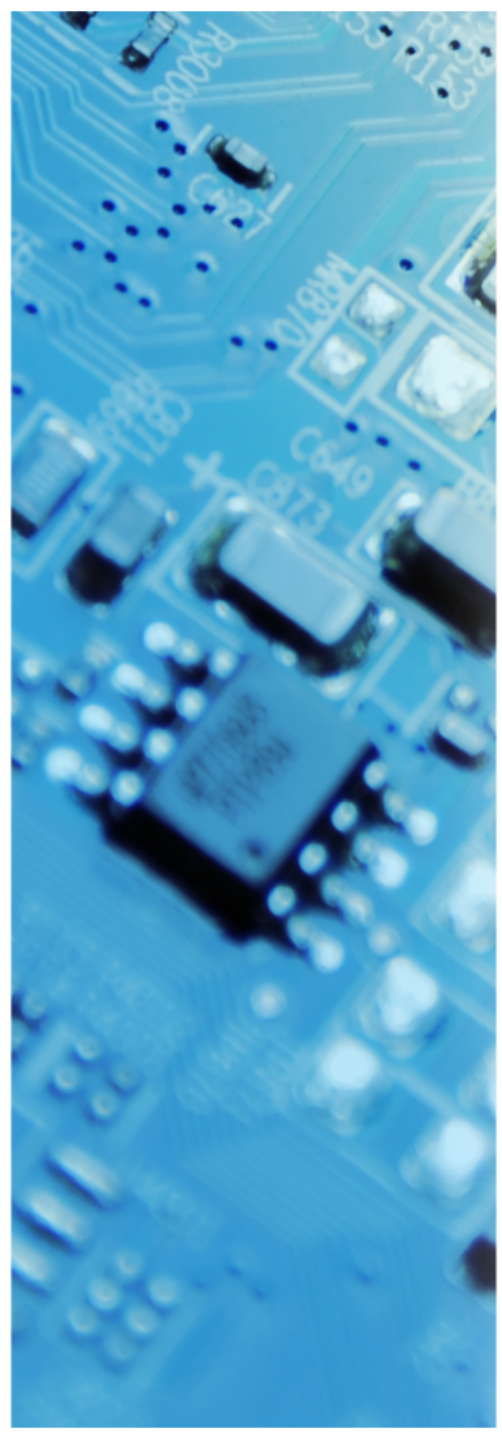
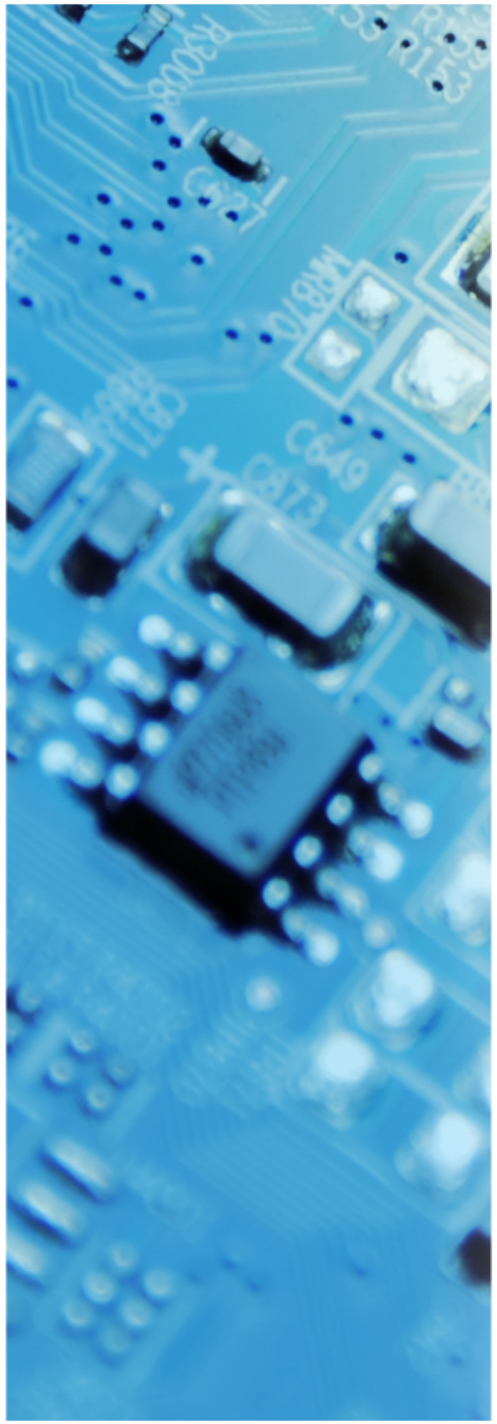


Funções de alta ordem
Recursividade
Ponteiros



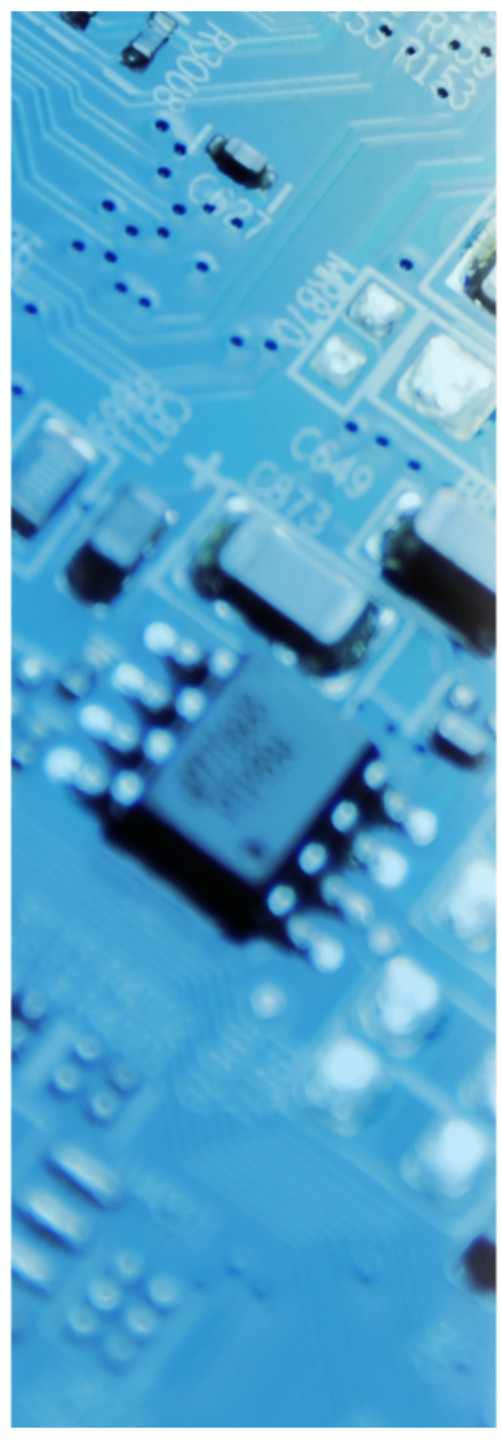
Funções de alta ordem

- Uma função é um objeto na memória, desta forma, é possível passar uma função como argumento em outra função.
- Python possui algumas funções especiais que utilizam funções como argumentos na sua execução.
- São conhecidos como funções de alta ordem.
- Todas as demais funções existentes ou criadas em Python são definidas como de primeira ordem.
- Funções de alta ordem são ferramentas utilizadas para aplicar conceitos de programação funcional.



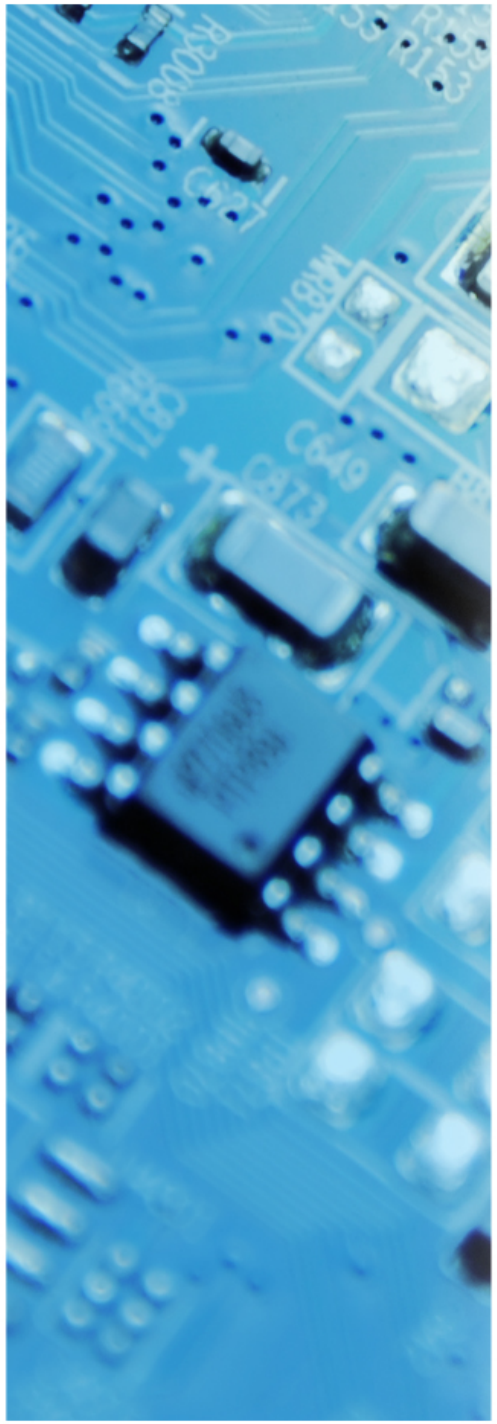
Funções de alta ordem

- Algumas funções de alta ordem fazem um mapeamento de uma função para um ou mais argumentos.
- Um exemplo é a função apply que está disponível em algumas versões da linguagem.
- Embora em versões mais recentes esta função não esteja disponível, é possível criar funções de alta ordem.
- Método apply está disponível na API Pandas e apresenta a mesma funcionalidade.



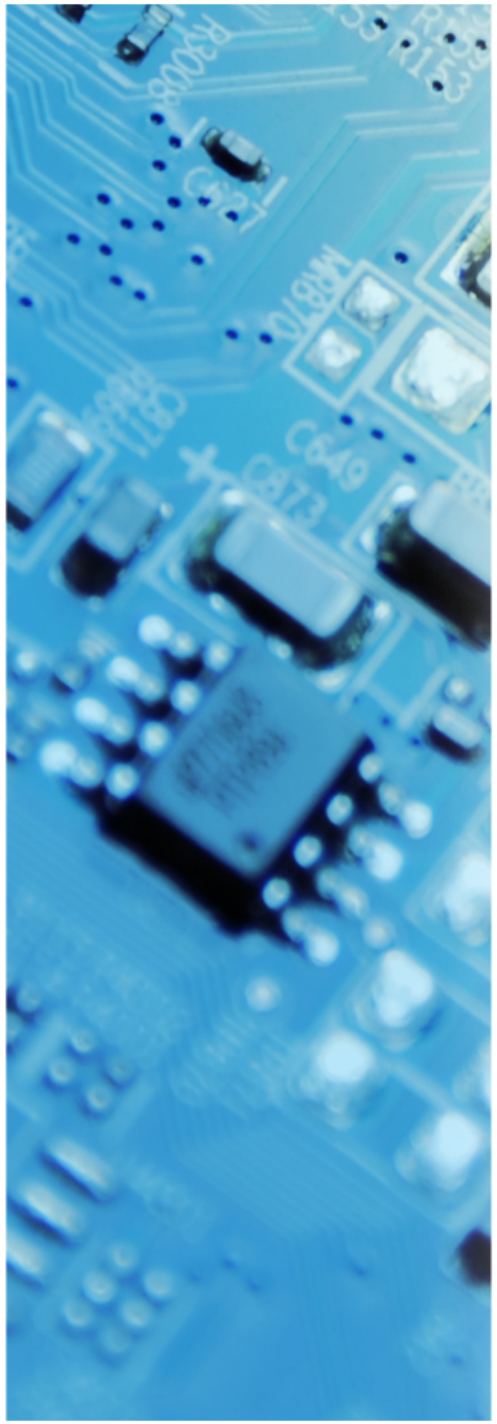
Funções de alta ordem

- A função de alta ordem map está disponível nativamente em Python.
- É utilizada para aplicar uma função a um conjunto de argumentos armazenados em um objeto iterável (lista, dicionário, ..).
- O retorno de map é uma referência para um objeto que pode ser convertido em uma lista ou outro tipo de estrutura.
- O conceito de mapear funções para valores é importante em diferentes contextos, especialmente em ciência de dados.



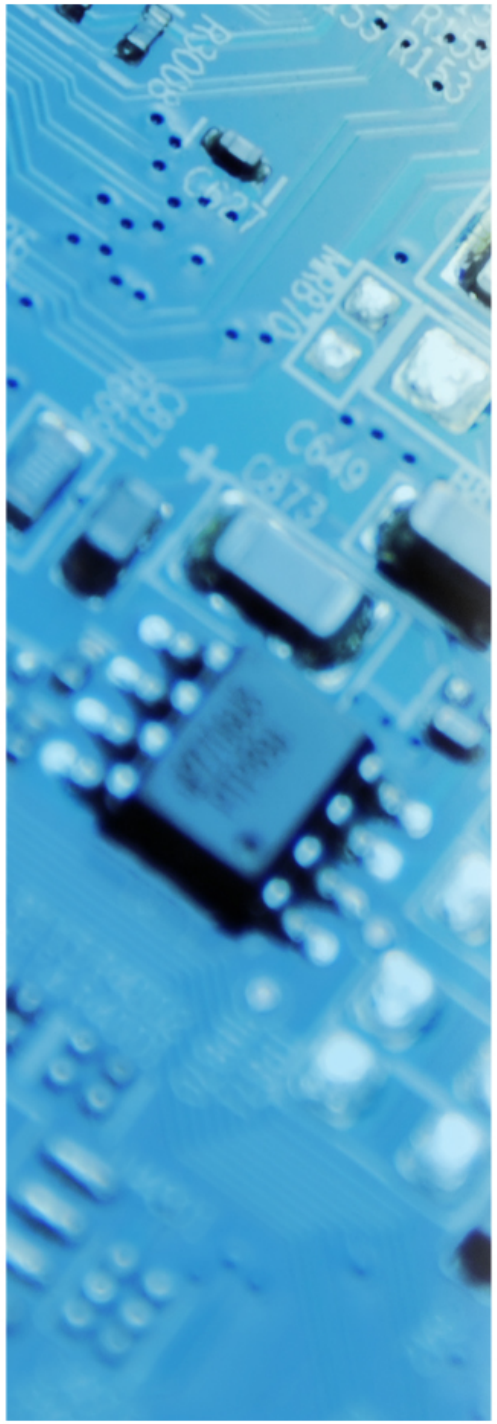
Funções de alta ordem

- A função filter é usada para aplicar uma seleção personalizada sobre um conjunto de argumentos.
- Os critérios da seleção podem ser definidos em uma função que será mapeada sobre os argumentos.
- Serão retornados somente os argumentos cujo teste condicional der positivo.
- Reduce por sua vez aplica uma função sobre um conjunto de argumentos retornando um único valor.
- Em aplicações que demandam alto processamento de dados e consultas distribuídas map/reduce são aplicadas em conjunto.



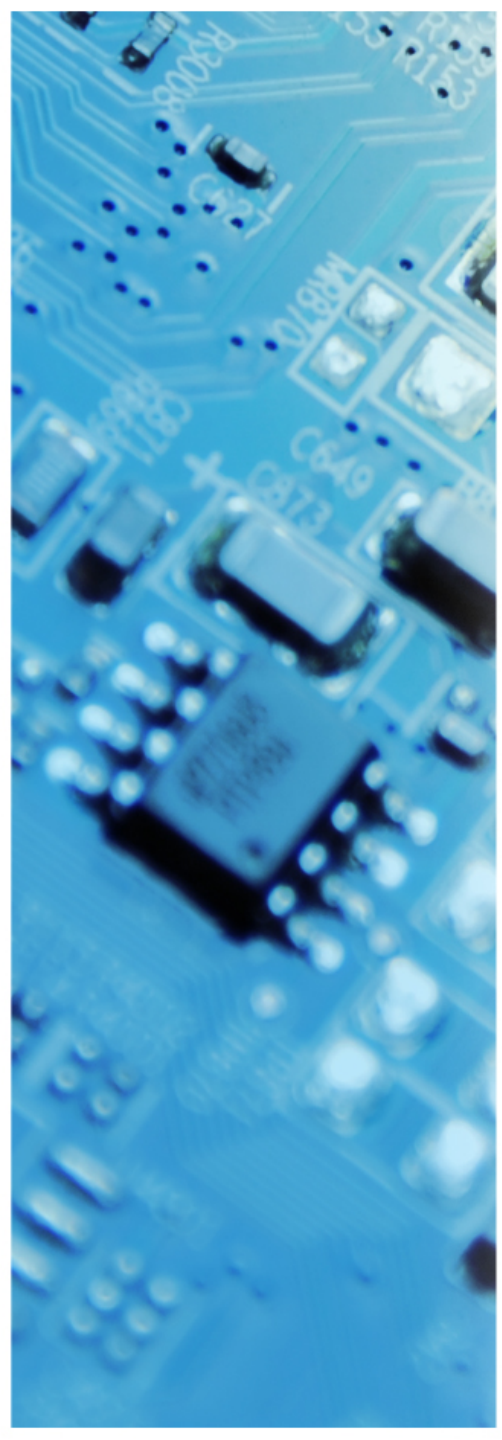
Execução de strings

- Em Python é possível executar o conteúdo armazenado em uma string.
- A função eval() avalia o conteúdo e se for uma expressão ou função, a mesma é executada.
- O resultado pode ser mostrado ou armazenado em uma variável.
- O retorno da execução é uma referência para um objeto.
- Este recurso permite criar códigos personalizados ou até mesmo situações nas quais o usuário indica a expressão que deseja executar.



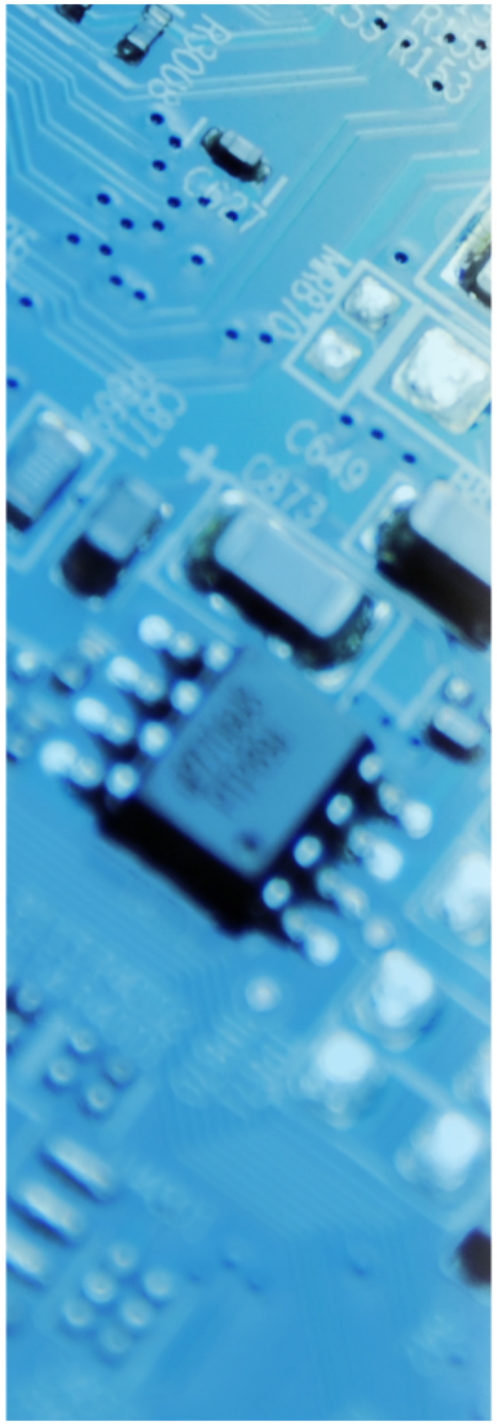
Recursividade

- Recursão é uma forma importante e elegante para resolver problemas.
- Um problema maior é decomposto em instâncias menores, com chamadas a uma mesma função.
- A solução final para o problema é gerada a partir dos resultados das diversas instâncias da solução.
- Cada instância do problema consiste em uma chamada a uma função.
- A diferença para uso tradicional de funções é que a função chama ela mesma.



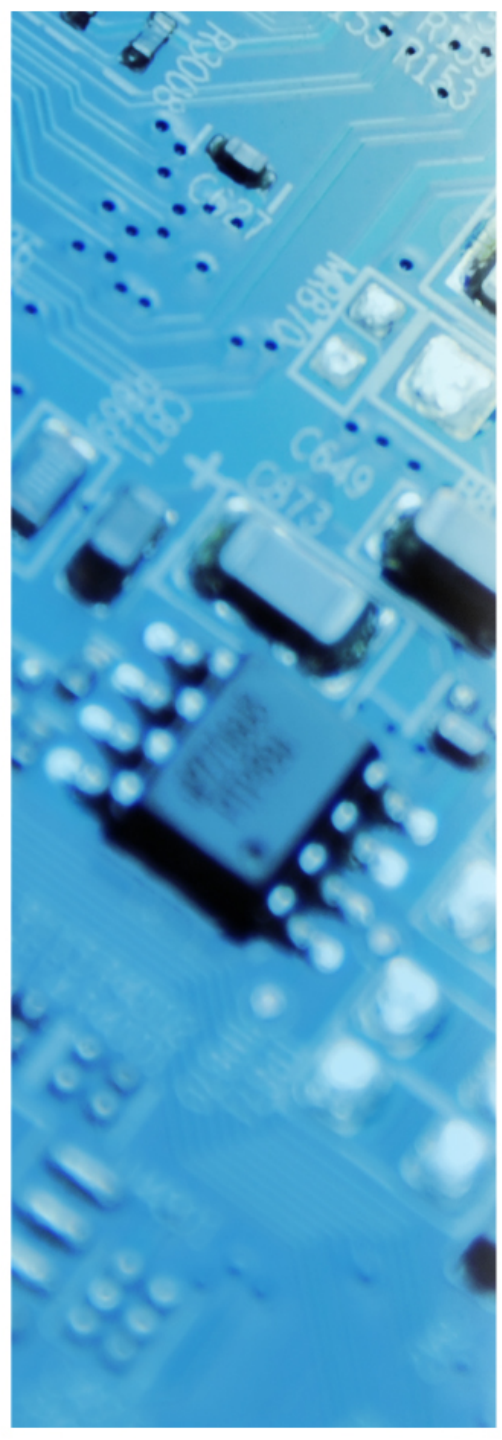
Recursividade

- Nem todos os problemas são resolvidos usando recursividade.
- Geralmente são problemas em que o processo de resolução é o mesmo para cada instância, porém as entradas estão em um nível abaixo.
- Exemplo: O fatorial de um número pode ser resolvido pela multiplicação dele pelo fatorial do número anterior.
- Cada número anterior é uma instância menor do problema, até chegar a 1.



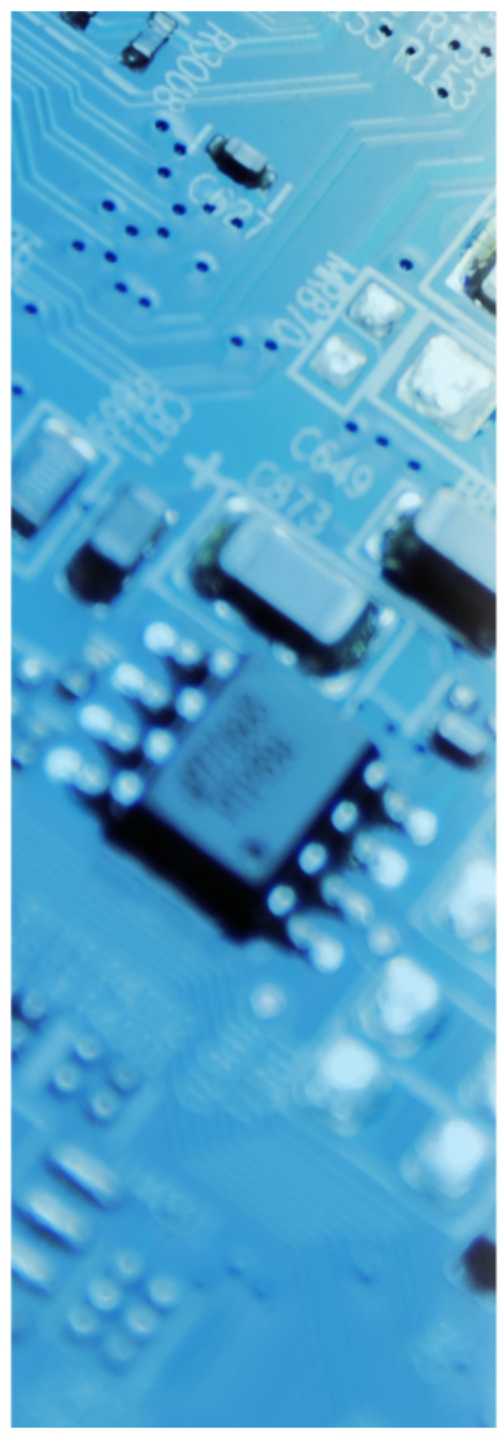
Recursividade

- Uma questão fundamental é que deve existir um critério de fim, ou seja, um momento em que a função para referir a si mesma.
- Neste momento ela retorna algum valor, sem fazer nova chamada.
- As diversas chamadas são inseridas em uma pilha na memória.
- Quando o primeiro retorno ocorre, as chamas vão sendo retiradas da pilha e vão retornando o valor até chegar à primeira chamada.



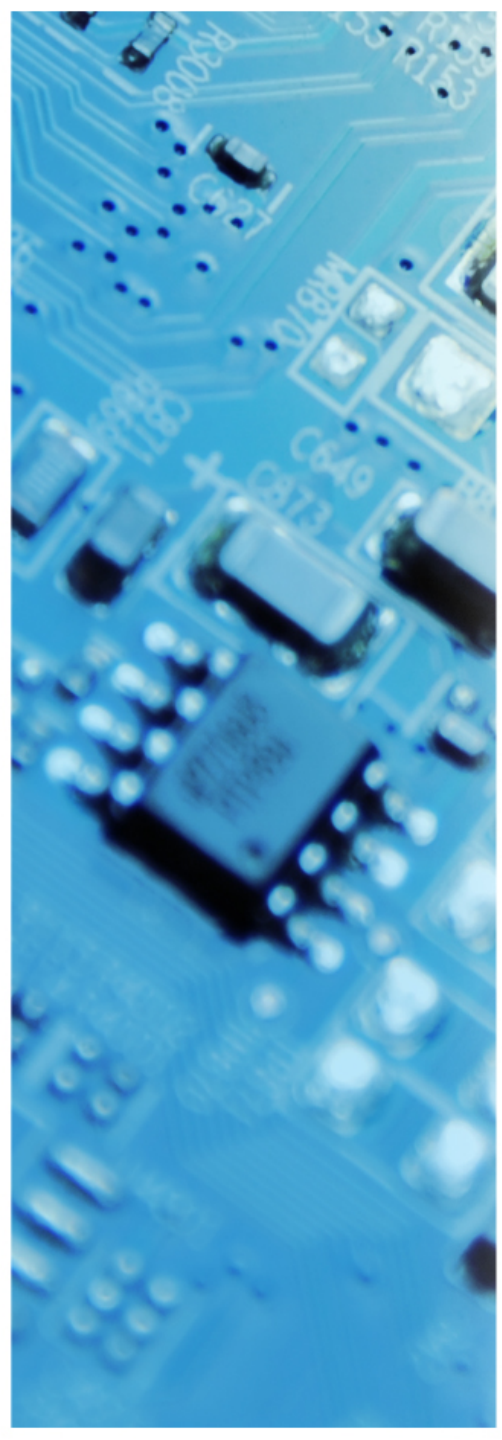
Recursividade

- Uma análise do problema vai indicar como irá ocorrer o retorno das chamadas.
- Fatores que podem indicar o fim da recursão:
  - Um valor encontrado, em uma estrutura, como árvore ou semelhante.
  - O problema está pequeno o suficiente para ser resolvido sem recursão.
  - Um nível máximo de recursão foi atingido.



Recursividade

- É importante ressaltar que a recursão nem sempre é a melhor alternativa.
- Em muitas situações, o uso de uma repetição pode ser computacionalmente mais eficiente.
- Em algumas situações o próprio interpretador expressa a solução interna como uma iteração, mesmo que tenha sido escrita com recursividade.
- O conhecimento sobre recursividade é essencial para qualquer programador, mesmo que o uso seja mais restrito.



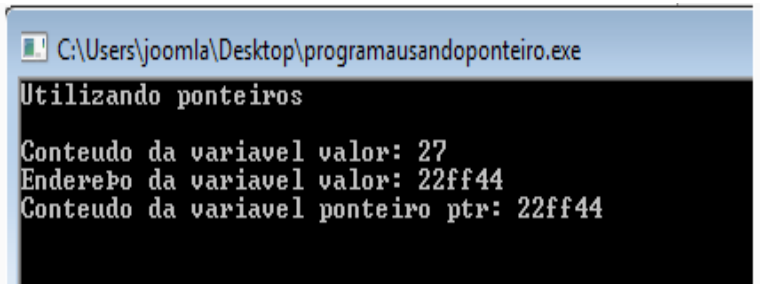
Ponteiros

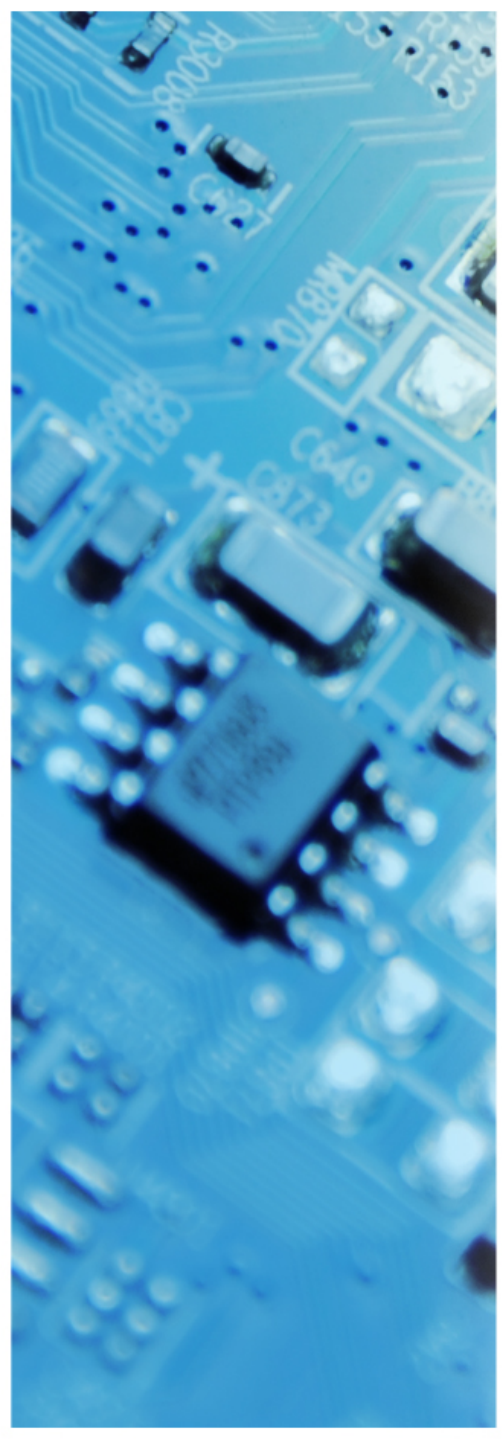
- Ponteiros podem ser definidos como variáveis que armazenam o endereço de um valor da memória.
- Em algumas linguagens, como C, é possível manipular ponteiros diretamente.
- O uso de ponteiros é comum para implementação de estruturas de dados, como listas, pilhas, deques, árvores, entre outras.
- Na linguagem python qualquer variável pode ser considerada um ponteiro, uma vez que possui uma referência para um objeto na memória.

Ponteiros - C

- Exemplo em http://linguagemc.com.br/ponteiros-em-c/.

```c
#include <stdio.h>
#include <conio.h>

int main(void)
{
  //valor é a variável que
  //será apontada pelo ponteiro
  int valor = 27;

  //declaração de variável ponteiro
  int *ptr;

  //atribuindo o endereço da variável valor ao ponteiro
  ptr = &valor;

  printf("Utilizando ponteiros\n\n");
  printf ("Conteudo da variavel valor: %d\n", valor);
  printf ("Endereço da variavel valor: %x \n", &valor);
  printf ("Conteudo da variavel ponteiro ptr: %x", ptr);

  getch();
  return(0);
}
```





Desafio

- Escreva um programa que crie uma lista com valores numéricos inteiros.
- A seguir solicite ao usuário que informe um valor.
- Escreva funções que retornem o número de valores da lista que são maiores do que o valor informado.
- Uma das funções deve ser recursiva e a outra deve utilizar uma instrução de repetição.
- Para testar crie uma lista com muitos valores e execute as duas funções para verificar se existe diferença clara de performance.
- Teste usando o colab e execute as funções com %timeit