

# Sobrecarga

Sobrecarga de Métodos  
Sobrecarga de Constructores

# Sumário

- Sobrecarga de métodos
- Sobrecarga de construtores
- Valor default
- \*args
- \*\*kwargs

# Sobrecarga de Métodos

- É a possibilidade de existência de **múltiplas versões** do mesmo método dentro da mesma classe, desde que possuam **assinaturas diferentes**.
  - as listas de tipos de seus parâmetros formais
- Possibilita a construção de classes versáteis, na qual a realização de certas operações admitam varias formas.

# **SOBRECARGA EM C++**

# Sobrecarga de Métodos

- Diferentes números de parâmetros
  - int **soma**(int a) { retorna a+a; }
  - int **soma**(int a, int b) { return a+b; }
  - int **soma**(int a, int b, int c) { return a+b+c; }

# Sobrecarga de Métodos

- Diferentes tipos de parâmetros
  - int **soma**(int a, int b) { return a+b; }
  - double **soma**(int a, double b) { return a+b; }
  - double **soma**(double a, double b) { return a+b; }

# Sobrecarga de Construtores

- Da mesma forma que métodos, construtores podem ser sobrecarregados.

```
Contador() { this->num=0; }
```

```
Contador(int num) { this->num = num; }
```

```
int main() {
```

```
    Contador c1, c2(9);
```

```
    ...
```

**VALOR DEFAULT**



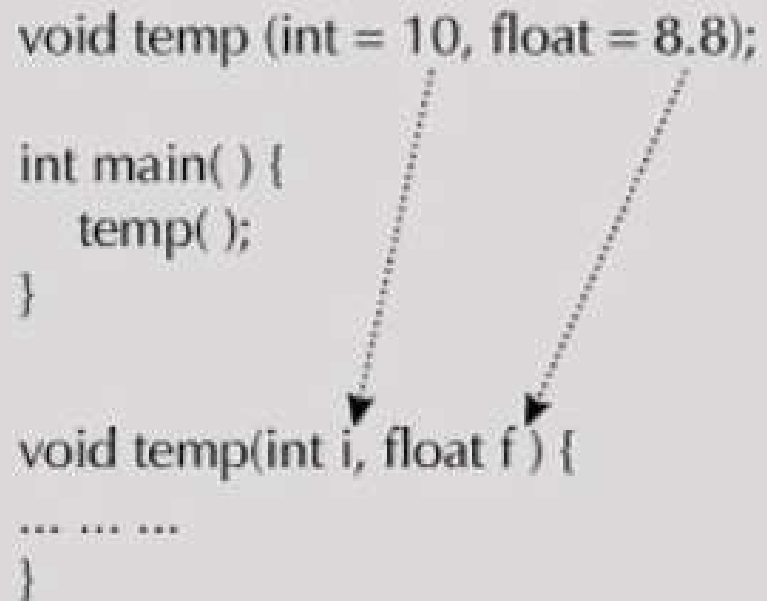
# Valor Default – Argumentos

- Em C++, você pode prover valores default (padrão) para parâmetros de funções.
- Se o argumento não for passado para a função, então o valor padrão é usado.

# Valor Default

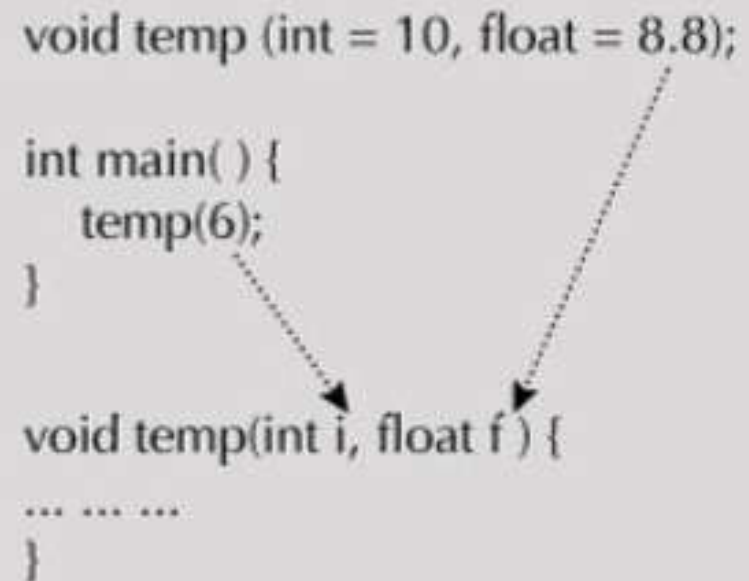
## Sem argumentos

```
void temp (int = 10, float = 8.8);  
  
int main( ) {  
    temp( );  
}  
  
void temp(int i, float f) {  
    ... ..  
}
```

A diagram illustrating default argument passing. Two dashed arrows originate from the default values '10' and '8.8' in the function declaration 'void temp (int = 10, float = 8.8);'. One arrow points to the parameter 'i' in the function definition 'void temp(int i, float f) {', and the other points to the parameter 'f'. This indicates that when the function is called without arguments, these default values are used.

## Primeiro argumento passado

```
void temp (int = 10, float = 8.8);  
  
int main( ) {  
    temp(6);  
}  
  
void temp(int i, float f) {  
    ... ..  
}
```

A diagram illustrating argument passing when the first argument is provided. A dashed arrow points from the value '6' in the function call 'temp(6);' to the parameter 'i' in the function definition 'void temp(int i, float f) {'. Another dashed arrow points from the default value '8.8' in the function declaration 'void temp (int = 10, float = 8.8);' to the parameter 'f' in the function definition. This shows that the provided argument '6' is used for 'i', and the default value '8.8' is used for 'f'.

# Valor Default

**Todos os argumentos passados**

```
void temp (int = 10, float = 8.8);
```

```
int main( ) {  
    temp(6, -2.3 );  
}
```

```
void temp(int i, float f ) {  
    ....  
}
```

The diagram consists of two dashed arrows. The first arrow originates from the number '6' in the function call 'temp(6, -2.3);' within the 'main' function and points to the parameter 'i' in the function definition 'void temp(int i, float f)'. The second arrow originates from the value '-2.3' in the same function call and points to the parameter 'f' in the function definition.

**Segundo argumento passado**

- IMPOSSÍVEL

# Problema na Sobrecarga

- Quando tem um parâmetro com valor default
- Dá ambiguidade
  - Ajusta(int a)
  - Ajusta(int a, int b = 9)
- São o mesmo para o compilador

**\*ARGS E \*\*KWARGS EM PYTHON**

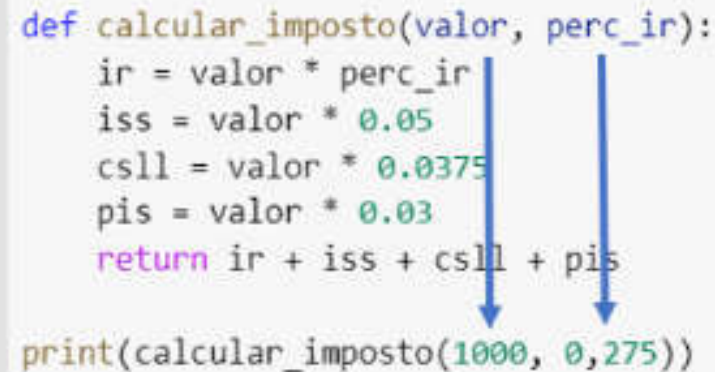
# Função de Exemplo

```
▶ def calcular_imposto(valor):  
    ir = valor * 0.275  
    iss = valor * 0.05  
    csll = valor * 0.0375  
    pis = valor * 0.03  
    return ir + iss + csll + pis  
  
print(calcular_imposto(1000))
```

392.5

# Argumentos de Posição

- O argumento de posição é intuitivo, temos duas variáveis, pela sequência sabemos a que variável cada valor pertence.



```
def calcular_imposto(valor, perc_ir):  
    ir = valor * perc_ir  
    iss = valor * 0.05  
    csll = valor * 0.0375  
    pis = valor * 0.03  
    return ir + iss + csll + pis  
  
print(calcular_imposto(1000, 0,275))
```

# Argumentos de Keyword

- Indicamos o nome da variável associada ao valor
  - valor e chave,
  - não importa a ordem que pusermos, o programa não vai se perder.

```
def calcular_imposto(valor, perc_ir):  
    ir = valor * perc_ir  
    iss = valor * 0.05  
    csll = valor * 0.0375  
    pis = valor * 0.03  
    return ir + iss + csll + pis  
  
print(calcular_imposto(perc_ir=0,275 , valor=1000))
```



# Argumentos de Keyword

- Não importa a quantidade de argumentos contanto que os de posição venham antes.

```
print(calcular_imposto(0,275 , valor=1000))
```



```
print(calcular_imposto(perc_ir=0,275 , 1000))
```



# Parâmetro \*ARGS

- \*ARGS substitui uma sequencia de valores
- O primeiro valor será somente 1000, já o segundo será quantos valores quisermos atribuir ao args.

```
def calcular_imposto(valor, *args):  
    total_imposto = 0  
    print(args)  
    return total_imposto  
print (calcular_imposto(1000, 0,25, 0,15, 15, 10))  
  
(0, 25, 0, 15, 15, 10)  
0
```

# Parâmetro \*ARGS

- \*ARGS substitui uma sequencia de valores
- O primeiro valor será somente 1000, já o segundo será quantos valores quisermos atribuir ao args.

```
def calcular_imposto(valor, *args):  
    total_imposto = 0  
    for item in args:  
        total_imposto += valor * item  
    return total_imposto  
  
print(calcular_imposto(1000, 0.275, 0.05, 0.0375, 0.03))
```

392.5

# Parâmetro **\*\*Kwargs**

- **kwargs** = KeyWord args
- A diferença da estrutura no código é que:
  - no **\*args** usamos 1 asterisco e não damos o nome ao parâmetro
  - no **\*\*kwargs** usamos 2 asteriscos e damos nome ao parâmetro.
- O resultado do Kwargs é um dicionário com todos os parâmetros nele

# Parâmetro **\*\*Kwargs**

```
def calcular_imposto(valor, **kwargs):  
    total_imposto = 0  
    print(kwargs)  
    return total_imposto  
  
print(calcular_imposto(1000, perc_ir=0.275, perc_iss=0.05, perc_csll=0.0375, perc_is=0.03))  
{'perc_ir': 0.275, 'perc_iss': 0.05, 'perc_csll': 0.0375, 'perc_is': 0.03}  
0
```

# Parâmetro \*\*KWARGS

```
def calcular_imposto_trimestral(valor, **kwargs):
    total_imposto = 0
    print(kwargs)
    if "perc_ir" in kwargs:
        total_imposto += valor * kwargs['perc_ir']
    if "perc_csll" in kwargs:
        total_imposto += valor * kwargs['perc_csll']
    return total_imposto

print(calcular_imposto_trimestral(1000, perc_iss=0.05, perc_pis=0.03, perc_ir=0.275, perc_csll=0.0375))

{'perc_iss': 0.05, 'perc_pis': 0.03, 'perc_ir': 0.275, 'perc_csll': 0.0375}
312.5
```

# **SOBRECARGA EM PYTHON**

# Sobrecarga m Python

- Em Python, a sobrecarga de métodos não é tratada da mesma forma que em algumas outras linguagens de programação
- Em Python, a sobrecarga de métodos é realizada de uma maneira mais flexível e dinâmica, geralmente utilizando
  - argumentos padrão
  - `*args` e `**kwargs` para aceitar um número variável de argumentos.



# Exemplo – Valor default

```
class Exemplo:
    def metodo(self, a=10, b=20, c=30):
        print(f'a={a}, b={b}, c={c}')

# Criando uma instância da classe Exemplo
exemplo = Exemplo()

# Chamadas de método com diferentes números de argumentos
exemplo.metodo()
exemplo.metodo(5)
exemplo.metodo(5, 15)
exemplo.metodo(5, 15, 25)
```

# Parâmetros – Número indefinido

- Podemos receber um número indefinido de parâmetros num métodos
- Isso é feito por meio de uma **tupla**
  - ()
  - (1,2)
  - (1,2,3,4)

# Exemplo – Número de parâmetros

```
class Exemplo:
    def metodo(self, *args):
        if len(args) == 0:
            print("Método sem argumentos")
        elif len(args) == 1:
            print("Método com um argumento:", args[0])
        else:
            print("Método com vários argumentos:", args)

# Criando uma instância da classe Exemplo
exemplo = Exemplo()

# Chamadas de método com diferentes números de argumentos
exemplo.metodo()
exemplo.metodo(10)
exemplo.metodo(10, 20, 30)
```

# Parâmetros – Vários Tipos

- Em Python, não é possível definir métodos com a mesma assinatura (ou seja, o mesmo nome e número de parâmetros) mas com tipos de parâmetros diferentes
- No entanto, você pode simular um comportamento semelhante verificando manualmente os tipos dos parâmetros dentro do método.
  - Use a função **isinstance()** para verificar se os argumentos fornecidos têm os tipos esperados.

# Exemplo – Tipos de parâmetros

```
class Exemplo:
    def metodo(self, arg):
        if isinstance(arg, int):
            print("O argumento é um inteiro:", arg)
        elif isinstance(arg, str):
            print("O argumento é uma string:", arg)
        else:
            print("Tipo de argumento não suportado:", type(arg))

# Criando uma instância da classe Exemplo
exemplo = Exemplo()

# Chamadas de método com diferentes tipos de argumentos
exemplo.metodo(10)
exemplo.metodo("hello")
exemplo.metodo(3.14)
```

# **EXERCÍCIOS**

# Exercício 1

- Para a classe Caixa, crie 3 formas para o método **ajustaMedidas()**, com:
  - 2 parâmetros: peso e empilhamento
  - 3 parâmetros: altura, largura e profundidade
  - 5 parâmetros: todos eles

```
double altura, largura, profundidade;  
double peso;  
int empilhamento;
```

## Exercício 2

- Para a classe Data, crie três construtores diferentes:
  - um com 3 parâmetros: dia, mês e ano
  - um sem parâmetros: que inicia com a data de hoje
  - um com 2 parâmetros: mês e ano
    - o dia passa a ser o primeiro dia do mês



## Exercício 3

- Crie uma classe chamada Ponto.
  - com 2 atributos: X e Y
- Crie dois construtores para esta classe:
  - um, sem parâmetros, que inicializa o ponto com 0,0
  - outro com 2 parâmetros
- Crie dois métodos chamados distancia:
  - um que recebe dois valores reais: X e Y
  - um que recebe outro Ponto