Introdução à programação em Python: funções

Neste notebook você vai aprender como fazer para reaproveitar trechos de código do seu programa.

Um aspecto importante na resolução de um problema complexo é conseguir dividi-lo em subproblemas menores.

Funções são estruturas que agrupam um conjunto de comandos, que são executados quando a função é chamada.

Nós já usamos várias funções até agora nesse curso.

In [8]:
```
print("'print()' é uma função")
valor = input("'input()' é outra função. Digite um número inteiro positivo: ")
valor = int(valor) #int() é outra função
# lista = list(range(valor)) #list() e range() são funções
lista = list(range(2,valor+2)) #list() e range() são funções
print(lista)
```

```
'print()' é uma função
'input()' é outra função. Digite um número inteiro positivo: 10
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

Por que usar funções?

Pode ser para evitar que os blocos do programa fiquem grandes demais e, por consequência, difíceis de ler e entender.

Ou para separar o programa em partes que possam ser logicamente compreendidas de forma isolada.

Ou para permitir o reaproveitamento de códigos, implementados por você ou por outros programadores.

E também para evitar que um trecho de código seja repetido várias vezes, evitando inconsistências e facilitando alterações.

Como criar funções?

A estrutura para **definir** uma função é a seguinte:

```
def nome_da_funcao(parametro1, parametro2, ...)
    comando usando ou não parametro1, parametro2, ...
    comando usando ou não parametro1, parametro2, ...
    etc.
    return

restante do programa
```

`parametrox` é o nome de uma variável que pode ser utilizada dentro da função. Uma função pode ter zero ou mais parâmetros.

`return` é um comando que indica o fim da função. Muitas vezes, funções produzem resultados que precisam ser devolvidos à quem as chamou. Assim, `return` é normalmente seguido de um valor/expressão.

Definir uma função não tem efeito nenhum em um programa (no sentido de que se houver, por exemplo, um comando `print()` dentro da função, nada será impresso se você apenas definiu a função.

Para de fato executar os comandos que estão dentros da função, você precisa fazer uma **chamada** à mesma:
`nome_da_funcao(argumento1, argumento2, ...)`

Você deve passar um valor para cada um dos parâmetros que aparecem na definição da função e na mesma ordem. Esses valores são chamados de argumentos durante a chamada.

É comum se dizer que uma função "recebe" um ou mais valores e "retorna" um resultado.

Importante: Uma função só pode ser chamada **depois** de sua definição.

Veja os exemplos a seguir para entender melhor esses conceitos.

Exemplo 1

A seguir temos a definição de uma função que tem um único parâmetro. Logo em seguida temos um exemplo de seu uso.

In [1]:
```
# A seguir temos a definição da função quadrado()
# Ela recebe um único parâmetro de nome x
# Dentro dela podemos usar x como uma variável normal
def quadrado(x):
    return x**2

# Note como a execução desse código não gerou saída alguma
```

In [2]:
```
valor = int(input("Digite um número qualquer: "))
# aqui estamos fazendo uma chamada à função quadrado()
# devemos passar um valor específico para o parâmetro que ela espera receber
# como ela retorna um valor, escolhemos salvá-lo em uma outra variável, para poder usá-lo depois
y = quadrado(valor)
print("O quadrado do número digitado é",y)

# Não há necessidade de salvar o valor em uma variável, no entanto
# Você pode usar a chamada da função diretamente:
print("O quadrado do número digitado é:", quadrado(valor))
print(quadrado(7))

# E também pode usar em expressões
soma = quadrado(5) + quadrado(10)
print(soma)
```

```
Digite um número qualquer: 2
O quadrado do número digitado é 4
O quadrado do número digitado é: 4
49
125
```

O que de fato ocorre quando você faz uma chamada a uma função?

Uma chamada a uma função é, essencialmente, uma cópia dos comandos que estão dentro da definição da função substituindo os valores dos parâmetros pelos valores que são passados na chamada.

Vamos verificar o que é feito quando o interpretador vê o comando `soma = quadrado(5) + quadrado(10)` do exemplo anterior.

Primeiro, como em qualquer comando de atribuição, ele precisa descobrir qual é o valor da expressão `quadrado(5) + quadrado(10)` para só então colocar esse valor na variável `soma`.

`quadrado(5)` é uma chamada de função, então o interpretador vai para a definição dessa função, coloca o valor 5 no parâmetro `x` e executa todos os comandos que estão dentro da função. No caso, apenas o comando `return x**2` é feito. Essencialmente então, `quadrado(5)` é substituído por aquela expressão que está depois do comando `return`, ou seja, `25`.

O programa volta então ao ponto onde havia parado, tendo agora a expressão `25 + quadrado(10)`. E `quadrado(10)` também é uma chamada de função, que por sua vez também é substituída pelo valor que está depois do comando `return`. Temos então `25 + 100`, que são somados e por fim atribuídos à variável `soma`.

Exemplo 2

Um coeficiente binomial é indexado por duas variáveis n e k, com $0 \leq k \leq n$, e é escrito da forma $\binom{n}{k}$. Ele é definido da seguinte forma:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

onde $x!$ denota o fatorial de x.

O programa a seguir calcula o coeficiente binomial de dois números lidos na entrada sem auxílio de funções.

In [4]:
```
k = int(input("Digite um número inteiro positivo (k): "))
n = int(input("Digite um número inteiro positivo maior do que o anterior (n): "))

# calculando fatorial de n
fatn = 1
for i in range(1,n+1):
    fatn = fatn * i

# calculando fatorial de k
fatk = 1
for i in range(1,k+1):
    fatk = fatk * i

# calculando fatorial de (n-k)
fatnk = 1
for i in range(1,(n-k)+1):
    fatnk = fatnk * i

coeficiente = fatn / (fatk * fatnk)
print("O coeficiente binomial de k e n é", coeficiente)
```

```
Digite um número inteiro positivo (k): 5
Digite um número inteiro positivo maior do que o anterior (n): 3
O coeficiente binomial de k e n é 0.05
```

Repare como existe código repetido no programa acima!

E se você descobrisse que estava calculando o fatorial de um número de forma errada? Ou então descobrisse uma forma melhor de calcular o fatorial de um número? Teria que sair arrumando todos os outros trechos de código onde esse cálculo aparece.

Note a seguir como o programa é simplificado com o uso de funções.

In [6]:
```
def fatorial(x):
    fat = 1
    for i in range(1,x+1):
        fat = fat * i
    return fat

k = int(input("Digite um número inteiro positivo: "))
n = int(input("Digite um número inteiro positivo maior do que o anterior: "))
coeficiente = fatorial(n)/(fatorial(k) * fatorial(n-k))
print("O coeficiente binomial de k e n é", coeficiente)
```

```
Digite um número inteiro positivo: 10
Digite um número inteiro positivo maior do que o anterior: 11
O coeficiente binomial de k e n é 11.0
```

Variáveis locais

Parâmetros e variáveis que você utiliza dentro da definição da função são chamados **locais** porque fora da definição, você não tem acesso a eles.

Veja o exemplo a seguir.

In [14]:
```
def fatorial(x):
    fat = 1
    for i in range(1,x+1):
        fat = fat * i
    return fat

print(fat)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-14-2f2853d34942> in <module>()
      5     return fat
      6
----> 7 print(fat)

NameError: name 'fat' is not defined
```

Repare que o erro ocorre porque tentamos acessar uma variável `fat` que não foi definida no programa: ela é local à função apenas.

Por causa disso, não importa se o nome das variáveis que você usa dentro ou fora da função são os mesmos, porque elas são completamente diferentes. Veja o próximo exemplo.

In [7]:
```
def fatorial(n):
    fat = 1
    for i in range(1,n+1):
        fat = fat * i
    return fat

k = int(input("Digite um número inteiro positivo: "))
n = int(input("Digite um número inteiro positivo maior do que o anterior: "))
coeficiente = fatorial(n)/(fatorial(k) * fatorial(n-k))
print("O coeficiente binomial de k e n é", coeficiente)
```

```
Digite um número inteiro positivo: 4
Digite um número inteiro positivo maior do que o anterior: 6
O coeficiente binomial de k e n é 15.0
```

Exemplo 3

No exemplo anterior de cálculo de coeficiente binomial era bem óbvio onde havia repetição de código. E funções ajudam muito quando isso acontece.

Mas um outro motivo para utilizar funções é quando queremos separar o programa em partes que possam ser logicamente compreendidas de forma isolada. Veremos um exemplo desses agora.

O programa a seguir verifica se um dado número é primo.

In [15]:
```
numero = int(input("Digite um número inteiro positivo: "))

# se ele for par maior do que 2, já não é primo
if numero > 2 and numero%2 == 0:
    print("Não é primo")
else:
    # agora basta verificar se ele tem algum divisor que não seja o 1 e ele mesmo
    # precisamos verificar todos os candidatos a divisores entre 3 e o número
    # se algum for divisor do número, ele não é primo
    # note que só precisamos verificar os candidatos ímpares
    primo = True
    print(list(range(3,numero)))
    for i in range(3,numero):
        if numero%i == 0:
            print("Não é primo")
            primo = False
    # nesse ponto, se a variável primo ainda é verdadeira, é porque nunca entramos no if
    # então nenhum número divide o número dado. Logo, ele é primo
    if primo:
        print("É primo")
```

```
Digite um número inteiro positivo: 7
[3, 4, 5, 6]
É primo
```

Podemos separar esse programa entre a parte que verifica se um número é primo e a parte que conversa com o usuário criando uma função cujo único objetivo é dizer se um número é primo ou não.

E uma função que decide se um número é primo ou não é útil em várias outras aplicações.

In [16]:
```
# essa função verifica se x é primo
# retorna True se for verdadeiro  e False caso contrário
def eh_primo(x):
    if x > 2 and x % 2 == 0:
        return False
    # Não preciso usar o else aqui, porque se a condição do if foi verdadeira,
    # então o programa acabou assim que leu o comando return
    for i in range(3,numero):
        if numero%i == 0:
            return False
    # Da mesma forma, se o programa chegar aqui é porque o for acabou e ele nunca entrou no if
    # Porque se tivesse entrado no if, teria acabado por ter lido o comando return
    return True

numero = int(input("Digite um número inteiro positivo: "))
if eh_primo(numero):
    print("É primo")
else:
    print("Não é primo")
```

```
Digite um número inteiro positivo: 11
É primo
```

Faça você mesmo!

O programa a seguir calcula o valor de $\sum_{i=0}^{j} a^i$ para cada valor de j entre 1 e n onde n e a são valores dados pelo usuário.

Modifique-o para que ele utilize uma função que calcula $\sum_{i=0}^{j} a^i$.

In [13]:
```
def somatorio(a, j):
    total = 0
    for i in range (0,j+1):
        total = total + a**i
    return total

n = int(input("Digite o valor de n: "))
a = int(input("Digite o valor de a: "))

for j in range(1,n+1):
   # total = 0
    #for i in range(j+1):
      #  total = total + a**i
    # print("Para j =", j, "a soma é", total)
     print("Para j =", j, "a soma é", somatorio(a, j))
```

```
Digite o valor de n: 5
Digite o valor de a: 3
Para j = 1 a soma é 4
Para j = 2 a soma é 13
Para j = 3 a soma é 40
Para j = 4 a soma é 121
Para j = 5 a soma é 364
```

Simulado para a prova

As questões a seguir idealmente devem ser feitas sem consulta a outros materiais e sem discussão com os colegas.

Aproveite a oportunidade para testar como estão seus conhecimentos para a prova da semana que vem!

Preste muita atenção nos exemplos e entenda bem os problemas.

Questão 1

Escreva um programa que lê uma coordenada (x, y) e imprima como resposta o quadrante em que a coordenada (x, y) está.

Entrada: O programa deve receber dois números reais x e y.

Saída: A resposta consiste de uma única linha, contendo **apenas uma** das seguintes frases, indicando onde (x, y) está: `Primeiro quadrante`, `Segundo quadrante`, `Terceiro quadrante`, `Quarto quadrante`, `Eixo x`, `Eixo y` ou `Centro`.

Exemplos de ENTRADA	SAÍDA esperada
5 5	Primeiro quadrante
-5 5	Segundo quadrante
-5 -5	Terceiro quadrante
5 -5	Quarto quadrante
0 -5	Eixo y
5 0	Eixo x
0 0	Centro

In [ ]:

Questão 2

Escreva um programa para ler n números e imprimir quantos deles estão nos seguintes intervalos: $[0 \dots 25]$, $[26 \dots 50]$, $[51 \dots 75]$ e $[76 \dots 100]$.

Entrada: O programa deve receber inicialmente um inteiro n ($n \geq 0$). Em seguida, deve receber n números reais **quaisquer**.

Saída: A resposta consiste de 4 linhas, cada uma contendo um intervalo e a quantidade de números lidos contidos em cada intervalo, conforme o exemplo abaixo.

Exemplo de ENTRADA	SAÍDA esperada
10	Intervalo [0..25]: 3
2.0 61.5 -1.0 0.0 88.7	Intervalo [26..50]: 0
94.5 55.0 3.1415 25.5 75.0	Intervalo [51..75]: 3
	Intervalo [76..100]: 2

In [ ]:

Questão 3

Um jogador da Mega-Quadra, a sensação do momento, é supersticioso e só faz jogos em que o primeiro número do jogo é par, o segundo é ímpar, o terceiro é par e o quarto é ímpar. Um jogo é definido como sendo uma sequência crescente de quatro números inteiros sem repetição, com cada número podendo valer entre 1 e 20. Note que, pelas definições acima, `2 5 26 31` é um jogo (números em ordem crescente) válido para esse jogador (números pares e ímpares intercalando) enquanto que `26 31 2 57` e `2 1 4 3` nem são jogos. Faça um programa que imprima todas as possibilidades de jogos que este jogador supersticioso pode jogar.

Entrada: Esse programa não tem entrada.

Saída: A resposta consiste de várias linhas, cada uma contendo seis números inteiros separados por espaço que indicam um possível jogo.

Exemplo de parte da SAÍDA
2 3 4 5
2 3 4 7
2 3 4 9
...
2 3 4 19
2 3 6 7
2 3 6 9
2 3 6 11
...
...
14 17 18 19
16 17 18 19

In [ ]: