Decompor o problema

Calcular m!

```
mfat = 1
i = 2
while i <= m:
    mfat *= i
    i += 1
```

mfat
24

Rascunho

4 x 3 x 2 x 1
6

m -

mfat *= i # mfat = mfat * i

m ↓ 4

mfat * 1 2 3 4 5
1 2 6 24 24

i 2 3 4 5

Rascunho

4! = 1 x 2 x 3 x 4
5! = (1 x 2 x 3 x 4) x 5
21 3! 4! 5!

mfat *2
mfat *3
mfat *4

f(x) = |x|
f(π) = 0

f((x, y)) = x - y → definição

f(4, 3) → 4 - 3 = 1
f(2, 5) = 2 - 5 = -3

usar chamada

Exercício

Dado m e n calcular

$$\frac{m!}{n!\ (m-n)!}$$

```
m = int(input(...)) # 3
n  = int(input(...)) # 2
```

```
def fatorial(k):
    '''(int) -> int
       Retorna k!
    '''
    kfat = 1
    i = 2
    while i <= k:
        kfat *= i
        i += 1
    return kfat
```

```
# calcule m!
mfat = 1
i = 2
while i <= m:
    mfat *= i
    i += 1
```

```
k = m #entra
kfat = 1
i = 2
while i <= k:
    kfat *= i
    i += 1
mfat = kfat #sai
```

```
mfat = fatorial(m)
```

```
# calcule n!
nfat = 1
i = 2
while i <= n:
    nfat *= i
    i += 1
```

```
k = n #entra
kfat = 1
i = 2
while i <= k:
    kfat *= i
    i += 1
nfat = kfat #sai
```

```
nfat = fatorial(n)
```

```
# calcule (m-n)!
mnfat = 1
i = 2
while i <= m-n:
    mnfat *= i
    i += 1
```

```
k = m-n #entra
kfat = 1
i = 2
while i <= k:
    kfat *= i
    i += 1
mnfat = kfat #sai
```

```
mnfat = fatorial(m-n)
```

```
bin = mfat//(nfat*mnfat)

print(bin)
```

main

m	n	mfat	nfat	mnfat
3	2	6	2	1

bin
3

imprime 3

fatorial

k	kfat	i
3	1	2
	2	3
	6	4

retorna 6

fatorial

k	kfat	i
2	1	2
	2	3

retorna 2

fatorial

k	kfat	i
1	1	2

retorna 1

Anatomia de um programa em Python

```
def main():
    ''' função principal '''

    lista/bloco de comandos


 # declaração de funções

def f(...):
    # corpo da função f

     bloco de comandos


def g(...):
    # corpo da função g

     bloco de comandos

if __name__ == "__main__":
    main()
```

Declaração de funções

```
def "nome da função"("parâmetros"):
    '''  Comentários (opcional, recomendado)
    '''
    # corpo da função

     bloco de comandos
```

Variáveis criadas dentro de uma função são locais, isto é, só existem dentro da função.

Parâmetros são variáveis locais criadas na chamada da função

Após a execução do comando return a execução da função é abandonada.

```
return "expressão"
```

Uma função também pode ser void e nesse caso a função não precisa de return.

Solução

```
def main():
    m = int(input("Digite m: "))
    n  = int(input("Digite n: "))

    mfat   = fatorial(m)
    nfat   = fatorial(n)
    mnfat = fatorial(m-n)
    binomial = mfat // (nfat*mnfat)

    print(f"bin({m},{n})={binomial}")


def fatorial(k):
    '''(int) -> int
       Recebe um inteiro k >= 0 e retorna k!
    '''
    kfat = 1
    i = 2
    while i <= k:
        kfat *= i
        i += 1
    return kfat

if __name__ == "__main__":
    main() # chamada da função principal
```