

MAC 110 – Introdução à Ciência da Computação

Aula 20

Nelson Lago

BMAC – 2024



Previously on MAC110...

Módulos

modulinho.py

```
print("Em modulinho.py: meu nome é", __name__)
```

Módulos

modulinho.py

```
print("Em modulinho.py: meu nome é", __name__)
```

Em modulinho.py: meu nome é

Módulos

modulinho.py

```
print("Em modulinho.py: meu nome é", __name__)
```

Em modulinho.py: meu nome é __main__

Módulos

modulinho.py

```
print("Em modulinho.py: meu nome é", __name__)
```

Em modulinho.py: meu nome é __main__

programinha.py

```
import modulinho  
print("Em programinha.py: meu nome é", __name__)
```

Módulos

modulinho.py

```
print("Em modulinho.py: meu nome é", __name__)
```

Em modulinho.py: meu nome é __main__

programinha.py

```
import modulinho  
print("Em programinha.py: meu nome é", __name__)
```

Em modulinho.py: meu nome é

Módulos

modulinho.py

```
print("Em modulinho.py: meu nome é", __name__)
```

Em modulinho.py: meu nome é __main__

programinha.py

```
import modulinho  
print("Em programinha.py: meu nome é", __name__)
```

Em modulinho.py: meu nome é modulinho

Módulos

modulinho.py

```
print("Em modulinho.py: meu nome é", __name__)
```

Em modulinho.py: meu nome é __main__

programinha.py

```
import modulinho  
print("Em programinha.py: meu nome é", __name__)
```

Em modulinho.py: meu nome é modulinho

Em programinha.py: meu nome é

Módulos

modulinho.py

```
print("Em modulinho.py: meu nome é", __name__)
```

Em modulinho.py: meu nome é __main__

programinha.py

```
import modulinho  
print("Em programinha.py: meu nome é", __name__)
```

Em modulinho.py: meu nome é modulinho

Em programinha.py: meu nome é __main__

Um módulo simples

- Vamos representar um ponto P no plano cartesiano como uma lista com dois elementos: as coordenadas x e y

Um módulo simples

- **Vamos representar um ponto P no plano cartesiano como uma lista com dois elementos: as coordenadas x e y**
 - ▶ $P = [x, y]$

Um módulo simples

- **Vamos representar um ponto P no plano cartesiano como uma lista com dois elementos: as coordenadas x e y**
 - ▶ $P = [x, y]$
- **Faça uma função que recebe dois pontos e devolve a distância euclidiana entre eles**

Um módulo simples

- **Vamos representar um ponto P no plano cartesiano como uma lista com dois elementos: as coordenadas x e y**
 - ▶ $P = [x, y]$
- **Faça uma função que recebe dois pontos e devolve a distância euclidiana entre eles**

Um módulo simples

- Vamos representar um ponto P no plano cartesiano como uma lista com dois elementos: as coordenadas x e y
 - ▶ $P = [x, y]$
- Faça uma função que recebe dois pontos e devolve a distância euclidiana entre eles

```
def distância(um, outro):
```

Um módulo simples

- Vamos representar um ponto P no plano cartesiano como uma lista com dois elementos: as coordenadas x e y
 - ▶ $P = [x, y]$
- Faça uma função que recebe dois pontos e devolve a distância euclidiana entre eles

```
def distância(um, outro):  
    dx = um[0] - outro[0]  
    dy = um[1] - outro[1]
```


Um módulo simples

- Vamos representar um ponto P no plano cartesiano como uma lista com dois elementos: as coordenadas x e y
 - ▶ $P = [x, y]$
- Faça uma função que recebe dois pontos e devolve a distância euclidiana entre eles

```
def distância(um, outro):  
    dx = um[0] - outro[0]  
    dy = um[1] - outro[1]  
    return math.sqrt(dx**2 + dy**2)
```

Um módulo simples

- Vamos representar um ponto P no plano cartesiano como uma lista com dois elementos: as coordenadas x e y
 - ▶ $P = [x, y]$
- Faça uma função que recebe dois pontos e devolve a distância euclidiana entre eles

```
import math
def distância(um, outro):
    dx = um[0] - outro[0]
    dy = um[1] - outro[1]
    return math.sqrt(dx**2 + dy**2)
```

Um módulo simples

- Vamos representar um polígono como uma lista de pontos representando seus vértices consecutivos

Um módulo simples

- **Vamos representar um polígono como uma lista de pontos representando seus vértices consecutivos**
 - ▶ Polígono = [P1, P2, P3, ..., Pn]

Um módulo simples

- **Vamos representar um polígono como uma lista de pontos representando seus vértices consecutivos**
 - ▶ Polígono = [P1, P2, P3, ..., Pn]
- **Usando a função `distância()`, faça uma função que recebe um polígono e devolve seu perímetro**

Um módulo simples

- **Vamos representar um polígono como uma lista de pontos representando seus vértices consecutivos**
 - ▶ Polígono = [P1, P2, P3, ..., Pn]
- **Usando a função `distância()`, faça uma função que recebe um polígono e devolve seu perímetro**

Um módulo simples

- **Vamos representar um polígono como uma lista de pontos representando seus vértices consecutivos**
 - ▶ Polígono = [P1, P2, P3, ..., Pn]
- **Usando a função `distância()`, faça uma função que recebe um polígono e devolve seu perímetro**

```
def perímetro(polígono):
```

Um módulo simples

- Vamos representar um polígono como uma lista de pontos representando seus vértices consecutivos
 - Polígono = [P1, P2, P3, ..., Pn]
- Usando a função `distância()`, faça uma função que recebe um polígono e devolve seu perímetro

```
def perímetro(polígono):  
    result = 0  
  
    return result
```


Um módulo simples

- **Vamos representar um polígono como uma lista de pontos representando seus vértices consecutivos**
 - Polígono = [P1, P2, P3, ..., Pn]
- **Usando a função `distância()`, faça uma função que recebe um polígono e devolve seu perímetro**

```
def perímetro(polígono):  
    result = 0  
  
    for pt in polígono:  
  
    return result
```

Um módulo simples

- **Vamos representar um polígono como uma lista de pontos representando seus vértices consecutivos**
 - ▶ Polígono = [P1, P2, P3, ..., Pn]
- **Usando a função `distância()`, faça uma função que recebe um polígono e devolve seu perímetro**

```
def perímetro(polígono):  
    result = 0  
  
    for pt in polígono:  
        result += distância(pt_anterior, pt)  
  
    return result
```

Um módulo simples

- **Vamos representar um polígono como uma lista de pontos representando seus vértices consecutivos**
 - Polígono = [P1, P2, P3, ..., Pn]
- **Usando a função `distância()`, faça uma função que recebe um polígono e devolve seu perímetro**

```
def perímetro(polígono):  
    result = 0  
  
    for pt in polígono:  
        result += distância(pt_anterior, pt)  
        pt_anterior = pt  
    return result
```

Um módulo simples

- **Vamos representar um polígono como uma lista de pontos representando seus vértices consecutivos**
 - ▶ Polígono = [P1, P2, P3, ..., Pn]
- **Usando a função `distância()`, faça uma função que recebe um polígono e devolve seu perímetro**

```
def perímetro(polígono):  
    result = 0  
    pt_anterior = polígono[-1]  
    for pt in polígono:  
        result += distância(pt_anterior, pt)  
        pt_anterior = pt  
    return result
```

Um módulo simples

poligono.py

```
import math
def distância(um, outro):
    dx = um[0] - outro[0]
    dy = um[1] - outro[1]
    return math.sqrt(dx**2 + dy**2)

def perímetro(polígono):
    result = 0
    pt_anterior = polígono[-1]
    for pt in polígono:
        result += distância(pt_anterior, pt)
        pt_anterior = pt
    return result
```

Um módulo simples

ponto.py

```
import math
def distância(um, outro):
    dx = um[0] - outro[0]
    dy = um[1] - outro[1]
    return math.sqrt(dx**2 + dy**2)
```

poligono.py

```
def perímetro(polígono):
    result = 0
    pt_anterior = polígono[-1]
    for pt in polígono:
        result += distância(pt_anterior, pt)
        pt_anterior = pt
    return result
```

Um módulo simples

ponto.py

```
import math
def distância(um, outro):
    dx = um[0] - outro[0]
    dy = um[1] - outro[1]
    return math.sqrt(dx**2 + dy**2)
```

poligono.py

```
import ponto
def perímetro(polígono):
    result = 0
    pt_anterior = polígono[-1]
    for pt in polígono:
        result += distância(pt_anterior, pt)
        pt_anterior = pt
    return result
```

Um módulo simples

ponto.py

```
import math
def distância(um, outro):
    dx = um[0] - outro[0]
    dy = um[1] - outro[1]
    return math.sqrt(dx**2 + dy**2)
```

poligono.py

```
import ponto
def perímetro(polígono):
    result = 0
    pt_anterior = polígono[-1]
    for pt in polígono:
        result += ponto.distância(pt_anterior, pt)
        pt_anterior = pt
    return result
```


Um módulo simples

poligono.py

```
import ponto

def perimetro(poligono):
    result = 0
    pt_anterior = poligono[-1]
    for pt in poligono:
        result += ponto.distancia(pt_anterior, pt)
        pt_anterior = pt
    return result
```

Um módulo simples

poligono.py

```
import ponto
dist = ponto.distância
def perímetro(polígono):
    result = 0
    pt_anterior = polígono[-1]
    for pt in polígono:
        result += ponto.distância(pt_anterior, pt)
        pt_anterior = pt
    return result
```

Um módulo simples

poligono.py

```
import ponto
dist = ponto.distância
def perimetro(polígono):
    result = 0
    pt_anterior = polígono[-1]
    for pt in polígono:
        result += dist(pt_anterior, pt)
        pt_anterior = pt
    return result
```

Um módulo simples

```
import ponto
def perimetro(poligono):
    result = 0
    pt_anterior = poligono[-1]
    for pt in poligono:
        result += ponto.distancia(pt_anterior, pt)
        pt_anterior = pt
    return result
```

Um módulo simples

```
import ponto
def perimetro(polígono):
    result = 0
    pt_anterior = polígono[-1]
    for pt in polígono:
        result += ponto.distância(pt_anterior, pt)
        pt_anterior = pt
    return result
def main():
    polígono = []
    x = input("Digite a coordenada x (\"enter\" para sair): ")
    while len(x) > 0:
        y = int(input("Digite a coordenada y: "))
        polígono.append([int(x), y])
        x = input("Digite a coordenada x (\"enter\" para sair): ")
    print("O perímetro do polígono é", perimetro(polígono))
```

Um módulo simples

```
import ponto
def perímetro(polígono):
    result = 0
    pt_anterior = polígono[-1]
    for pt in polígono:
        result += ponto.distância(pt_anterior, pt)
        pt_anterior = pt
    return result
def main():
    polígono = []
    x = input("Digite a coordenada x (\"enter\" para sair): ")
    while len(x) > 0:
        y = int(input("Digite a coordenada y: "))
        polígono.append([int(x), y])
        x = input("Digite a coordenada x (\"enter\" para sair): ")
    print("O perímetro do polígono é", perímetro(polígono))
if __name__ == "__main__":
    main()
```

Quais testes você criaria para

Quais testes você criaria para

- A função **fatorial()**?

Quais testes você criaria para

- A função `fatorial()`?
- A função `éPrimo()`?

Quais testes você criaria para

- A função `fatorial()`?
- A função `éPrimo()`?
- Uma função que recebe uma lista e um valor e devolve o índice da primeira ocorrência do valor na lista ou -1 caso o valor não esteja presente na lista?

Exercício – testando `fatorial()`



Exercício – testando `fatorial()`

```
def fatorial(n):  
    fat = 1  
    while n > 1:  
        fat *= n  
        n -= 1  
    return fat
```

Exercício – testando `fatorial()`

```
import pytest
def fatorial(n):
    fat = 1
    while n > 1:
        fat *= n
        n -= 1
    return fat
```

Exercício – testando `fatorial()`

```
import pytest
def fatorial(n):
    fat = 1
    while n > 1:
        fat *= n
        n -= 1
    return fat

def test_fat(entrada, saída):
```

Exercício – testando fatorial()

```
import pytest

def fatorial(n):
    fat = 1
    while n > 1:
        fat *= n
        n -= 1
    return fat

def test_fat(entrada, saída):
    assert saída == fatorial(entrada)
```

Exercício – testando fatorial()

```
import pytest
def fatorial(n):
    fat = 1
    while n > 1:
        fat *= n
        n -= 1
    return fat

@pytest.mark.parametrize("entrada, saída", [
def test_fat(entrada, saída):
    assert saída == fatorial(entrada)
```


Exercício – testando fatorial()

```
import pytest
def fatorial(n):
    fat = 1
    while n > 1:
        fat *= n
        n -= 1
    return fat

@pytest.mark.parametrize("entrada, saída", [(5, 120),
])
def test_fat(entrada, saída):
    assert saída == fatorial(entrada)
```

Exercício – testando fatorial()

```
import pytest
def fatorial(n):
    fat = 1
    while n > 1:
        fat *= n
        n -= 1
    return fat

@pytest.mark.parametrize("entrada, saída", [(5, 120), (2, 2),
])
def test_fat(entrada, saída):
    assert saída == fatorial(entrada)
```

Exercício – testando fatorial()

```
import pytest
def fatorial(n):
    fat = 1
    while n > 1:
        fat *= n
        n -= 1
    return fat

@pytest.mark.parametrize("entrada, saída", [(5, 120), (2, 2), (1, 1),      ])
def test_fat(entrada, saída):
    assert saída == fatorial(entrada)
```

Exercício – testando fatorial()

```
import pytest

def fatorial(n):
    fat = 1
    while n > 1:
        fat *= n
        n -= 1
    return fat

@pytest.mark.parametrize("entrada, saída", [(5, 120), (2, 2), (1, 1), (0, 1)])
def test_fat(entrada, saída):
    assert saída == fatorial(entrada)
```

Exercício – testando éPrimo()

Exercício – testando éPrimo()

```
def éPrimo(x):  
    if x < 2:  
        return False  
    divisor = x - 1  
    while divisor >= 2:  
        if x % divisor == 0:  
            return False  
        divisor -= 1  
    return True
```

Exercício – testando éPrimo()

```
import pytest
def éPrimo(x):
    if x < 2:
        return False
    divisor = x - 1
    while divisor >= 2:
        if x % divisor == 0:
            return False
        divisor -= 1
    return True
```

Exercício – testando éPrimo()

```
import pytest
def éPrimo(x):
    if x < 2:
        return False
    divisor = x - 1
    while divisor >= 2:
        if x % divisor == 0:
            return False
        divisor -= 1
    return True

def test_éPrimo(entrada, saída):
```


Exercício – testando éPrimo()

```
import pytest
def éPrimo(x):
    if x < 2:
        return False
    divisor = x - 1
    while divisor >= 2:
        if x % divisor == 0:
            return False
        divisor -= 1
    return True

def test_éPrimo(entrada, saída):
    assert saída == éPrimo(entrada)
```

Exercício – testando éPrimo()

```
import pytest
def éPrimo(x):
    if x < 2:
        return False
    divisor = x - 1
    while divisor >= 2:
        if x % divisor == 0:
            return False
        divisor -= 1
    return True

@pytest.mark.parametrize("entrada, saída", [

def test_éPrimo(entrada, saída):
    assert saída == éPrimo(entrada)
```

)

Exercício – testando éPrimo()

```
import pytest
def éPrimo(x):
    if x < 2:
        return False
    divisor = x - 1
    while divisor >= 2:
        if x % divisor == 0:
            return False
        divisor -= 1
    return True

@pytest.mark.parametrize("entrada, saída", [
    (7, True),
])

def test_éPrimo(entrada, saída):
    assert saída == éPrimo(entrada)
```

Exercício – testando éPrimo()

```
import pytest
def éPrimo(x):
    if x < 2:
        return False
    divisor = x - 1
    while divisor >= 2:
        if x % divisor == 0:
            return False
        divisor -= 1
    return True

@pytest.mark.parametrize("entrada, saída", [
    (7, True),
    (105, False)])
def test_éPrimo(entrada, saída):
    assert saída == éPrimo(entrada)
```

Exercício – testando éPrimo()

```
import pytest
def éPrimo(x):
    if x < 2:
        return False
    divisor = x - 1
    while divisor >= 2:
        if x % divisor == 0:
            return False
        divisor -= 1
    return True

@pytest.mark.parametrize("entrada, saída", [
    (3, True), (4, False), (5, True),
    (6, False), (7, True),
    (105, False)])

def test_éPrimo(entrada, saída):
    assert saída == éPrimo(entrada)
```

Exercício – testando éPrimo()

```
import pytest
def éPrimo(x):
    if x < 2:
        return False
    divisor = x - 1
    while divisor >= 2:
        if x % divisor == 0:
            return False
        divisor -= 1
    return True

@pytest.mark.parametrize("entrada, saída", [
    (3, True), (4, False), (5, True),
    (6, False), (7, True), (99, False),
    (100, False), (101, True), (102, False),
    (103, True), (104, False), (105, False)])
def test_éPrimo(entrada, saída):
    assert saída == éPrimo(entrada)
```

Exercício – testando éPrimo()

```
import pytest
def éPrimo(x):
    if x < 2:
        return False
    divisor = x - 1
    while divisor >= 2:
        if x % divisor == 0:
            return False
        divisor -= 1
    return True

@pytest.mark.parametrize("entrada, saída", [
    (2, True),
    (3, True), (4, False), (5, True),
    (6, False), (7, True), (99, False),
    (100, False), (101, True), (102, False),
    (103, True), (104, False), (105, False)])

def test_éPrimo(entrada, saída):
    assert saída == éPrimo(entrada)
```

Exercício – testando éPrimo()

```
import pytest
def éPrimo(x):
    if x < 2:
        return False
    divisor = x - 1
    while divisor >= 2:
        if x % divisor == 0:
            return False
        divisor -= 1
    return True

@pytest.mark.parametrize("entrada, saída", [(0, False), (1, False), (2, True),
                                             (3, True), (4, False), (5, True),
                                             (6, False), (7, True), (99, False),
                                             (100, False), (101, True), (102, False),
                                             (103, True), (104, False), (105, False)])

def test_éPrimo(entrada, saída):
    assert saída == éPrimo(entrada)
```


And now for something completely different

- **Existem alguns tipos diferentes de documentação**

- **Existem alguns tipos diferentes de documentação**
 - Documentação para o uso

- **Existem alguns tipos diferentes de documentação**
 - Documentação para o uso
 - » *Manuais*

- **Existem alguns tipos diferentes de documentação**
 - ▶ Documentação para o uso
 - » *Manuais*
 - » *Tutoriais*

- **Existem alguns tipos diferentes de documentação**
 - ▶ Documentação para o uso
 - » *Manuais*
 - » *Tutoriais*
 - » *Referência*

- **Existem alguns tipos diferentes de documentação**
 - ▶ Documentação para o uso
 - » *Manuais*
 - » *Tutoriais*
 - » *Referência*
 - ▶ Documentação para os desenvolvedores

- **Existem alguns tipos diferentes de documentação**
 - ▶ Documentação para o uso
 - » *Manuais*
 - » *Tutoriais*
 - » *Referência*
 - ▶ Documentação para os desenvolvedores
 - » *Descrição da arquitetura e suas motivações*

- **Existem alguns tipos diferentes de documentação**

- ▶ Documentação para o uso
 - » *Manuais*
 - » *Tutoriais*
 - » *Referência*
- ▶ Documentação para os desenvolvedores
 - » *Descrição da arquitetura e suas motivações*
 - » *Comentários*

- **Existem alguns tipos diferentes de documentação**
 - ▶ Documentação para o uso
 - » *Manuais*
 - » *Tutoriais*
 - » *Referência*
 - ▶ Documentação para os desenvolvedores
 - » *Descrição da arquitetura e suas motivações*
 - » *Comentários*
 - » *Descrição das interfaces*

- **Existem alguns tipos diferentes de documentação**
 - ▶ Documentação para o uso
 - » *Manuais*
 - » *Tutoriais*
 - » *Referência*
 - ▶ Documentação para os desenvolvedores
 - » *Descrição da arquitetura e suas motivações*
 - » *Comentários*
 - » *Descrição das **interfaces***
(o que cada função faz, quais parâmetros recebe, quais valores devolve etc.)

- **Descrição da arquitetura e suas motivações**

- **Descrição da arquitetura e suas motivações**
 - ▶ Visão de “alto nível”

- **Descrição da arquitetura e suas motivações**

- ▶ Visão de “alto nível”
- ▶ Importante para quem precisa entender o código para poder modificá-lo

- **Descrição da arquitetura e suas motivações**
 - ▶ Visão de “alto nível”
 - ▶ Importante para quem precisa entender o código para poder modificá-lo
- **Comentários**

- **Descrição da arquitetura e suas motivações**
 - ▶ Visão de “alto nível”
 - ▶ Importante para quem precisa entender o código para poder modificá-lo
- **Comentários**
 - ▶ Visão de “baixo nível” (trechos pequenos)

- **Descrição da arquitetura e suas motivações**

- ▶ Visão de “alto nível”
- ▶ Importante para quem precisa entender o código para poder modificá-lo

- **Comentários**

- ▶ Visão de “baixo nível” (trechos pequenos)
- ▶ Importantes para quem precisa entender o código para poder modificá-lo

- **Descrição da arquitetura e suas motivações**

- ▶ Visão de “alto nível”
- ▶ Importante para quem precisa entender o código para poder modificá-lo

- **Comentários**

- ▶ Visão de “baixo nível” (trechos pequenos)
- ▶ Importantes para quem precisa entender o código para poder modificá-lo
- ▶ Idealmente, raros, pois o próprio código deve ser claro o suficiente

- **Descrição da arquitetura e suas motivações**
 - ▶ Visão de “alto nível”
 - ▶ Importante para quem precisa entender o código para poder modificá-lo
- **Comentários**
 - ▶ Visão de “baixo nível” (trechos pequenos)
 - ▶ Importantes para quem precisa entender o código para poder modificá-lo
 - ▶ Idealmente, raros, pois o próprio código deve ser claro o suficiente
- **Descrições das interfaces**

- **Descrição da arquitetura e suas motivações**
 - ▶ Visão de “alto nível”
 - ▶ Importante para quem precisa entender o código para poder modificá-lo
- **Comentários**
 - ▶ Visão de “baixo nível” (trechos pequenos)
 - ▶ Importantes para quem precisa entender o código para poder modificá-lo
 - ▶ Idealmente, raros, pois o próprio código deve ser claro o suficiente
- **Descrições das interfaces**
 - ▶ Visão de “médio nível”

- **Descrição da arquitetura e suas motivações**
 - ▶ Visão de “alto nível”
 - ▶ Importante para quem precisa entender o código para poder modificá-lo
- **Comentários**
 - ▶ Visão de “baixo nível” (trechos pequenos)
 - ▶ Importantes para quem precisa entender o código para poder modificá-lo
 - ▶ Idealmente, raros, pois o próprio código deve ser claro o suficiente
- **Descrições das interfaces**
 - ▶ Visão de “médio nível”
 - ▶ Importantes para quem precisa entender o código para poder modificá-lo

- **Descrição da arquitetura e suas motivações**
 - ▶ Visão de “alto nível”
 - ▶ Importante para quem precisa entender o código para poder modificá-lo
- **Comentários**
 - ▶ Visão de “baixo nível” (trechos pequenos)
 - ▶ Importantes para quem precisa entender o código para poder modificá-lo
 - ▶ Idealmente, raros, pois o próprio código deve ser claro o suficiente
- **Descrições das interfaces**
 - ▶ Visão de “médio nível”
 - ▶ Importantes para quem precisa entender o código para poder modificá-lo
 - ▶ Importantes também para quem vai **usar** trechos do código (módulos) em outros projetos

- **Conforme o código evolui, a documentação deve ser modificada de acordo**

- **Conforme o código evolui, a documentação deve ser modificada de acordo**
- **MAS...**

- Conforme o código evolui, a documentação deve ser modificada de acordo
- **MAS...**
- Na prática, é muito fácil “esquecer” de atualizar a documentação

- **Conforme o código evolui, a documentação deve ser modificada de acordo**
- **MAS...**
- **Na prática, é muito fácil “esquecer” de atualizar a documentação**
 - ▶ E documentação desatualizada/incorreta pode ser pior que não haver documentação!

- **Descrição da arquitetura e suas motivações**

- **Descrição da arquitetura e suas motivações**
 - ▶ Mudanças são lentas e grandes, então as chances de a documentação ser atualizada são melhores

- **Descrição da arquitetura e suas motivações**
 - ▶ Mudanças são lentas e grandes, então as chances de a documentação ser atualizada são melhores
 - » *Especialmente se a documentação for concisa*

- **Descrição da arquitetura e suas motivações**
 - ▶ Mudanças são lentas e grandes, então as chances de a documentação ser atualizada são melhores
 - » *Especialmente se a documentação for concisa*
- **Comentários**

- **Descrição da arquitetura e suas motivações**

- ▶ Mudanças são lentas e grandes, então as chances de a documentação ser atualizada são melhores
 - » *Especialmente se a documentação for concisa*

- **Comentários**

- ▶ Se forem raros e concisos, têm alguma chance de serem atualizados

- **Descrição da arquitetura e suas motivações**

- ▶ Mudanças são lentas e grandes, então as chances de a documentação ser atualizada são melhores
 - » *Especialmente se a documentação for concisa*

- **Comentários**

- ▶ Se forem raros e concisos, têm alguma chance de serem atualizados
 - » *E, sendo raros, erros são menos graves*

- **Descrição da arquitetura e suas motivações**

- ▶ Mudanças são lentas e grandes, então as chances de a documentação ser atualizada são melhores
 - » *Especialmente se a documentação for concisa*

- **Comentários**

- ▶ Se forem raros e concisos, têm alguma chance de serem atualizados
 - » *E, sendo raros, erros são menos graves*

- **Descrições das interfaces**

- **Descrição da arquitetura e suas motivações**

- ▶ Mudanças são lentas e grandes, então as chances de a documentação ser atualizada são melhores
 - » *Especialmente se a documentação for concisa*

- **Comentários**

- ▶ Se forem raros e concisos, têm alguma chance de serem atualizados
 - » *E, sendo raros, erros são menos graves*

- **Descrições das interfaces**

- ▶ Com muitas funções, a documentação é extensa e trabalhosa 😞

- **Descrição da arquitetura e suas motivações**

- ▶ Mudanças são lentas e grandes, então as chances de a documentação ser atualizada são melhores
 - » *Especialmente se a documentação for concisa*

- **Comentários**

- ▶ Se forem raros e concisos, têm alguma chance de serem atualizados
 - » *E, sendo raros, erros são menos graves*

- **Descrições das interfaces**

- ▶ Com muitas funções, a documentação é extensa e trabalhosa 😞
- ▶ Podem ser parcialmente automatizadas 🎉

```
def éPrimo(x):  
  
    if x < 2:  
        return False  
    divisor = x - 1  
    while divisor >= 2:  
        if x % divisor == 0:  
            return False  
        divisor -= 1  
    return True
```

```
def éPrimo(x):  
    """Recebe um inteiro >=0 e devolve um booleano."""  
    if x < 2:  
        return False  
    divisor = x - 1  
    while divisor >= 2:  
        if x % divisor == 0:  
            return False  
        divisor -= 1  
    return True
```

```
def imprime_matriz(A):  
  
    for linha in A:  
        for col in linha:  
            print("{:4}".format(col), end="")  
        print()
```

```
def imprime_matriz(A):  
    """Recebe uma matriz (lista de listas) e a imprime no terminal.  
  
    """  
    for linha in A:  
        for col in linha:  
            print("{:4}".format(col), end="")  
        print()
```

```
def imprime_matriz(A):  
    """Recebe uma matriz (lista de listas) e a imprime no terminal.  
  
    Cada elemento da matriz pode ter até 3 dígitos/caracteres  
    """  
    for linha in A:  
        for col in linha:  
            print("{:4}".format(col), end="")  
        print()
```


Resumo (uma linha)

```
def imprime_matriz(A):  
    """Recebe uma matriz (lista de listas) e a imprime no terminal.  
  
    Cada elemento da matriz pode ter até 3 dígitos/caracteres  
    """  
    for linha in A:  
        for col in linha:  
            print("{:4}".format(col), end="")  
        print()
```

```
def imprime_matriz(A):  
    """Recebe uma matriz (lista de listas) e a imprime no terminal.  
  
    Cada elemento da matriz pode ter até 3 dígitos/caracteres  
    """  
    for linha in A:  
        for col in linha:  
            print("{:4}".format(col), end="")  
        print()
```

Resumo (uma linha)

Detalhes (várias linhas)

Docstrings

```
def imprime_matriz(A):
```

```
    """Recebe uma matriz (lista de listas) e a imprime no terminal.
```

```
    Cada elemento da matriz pode ter até 3 dígitos/caracteres
```

```
    """
```

```
    for linha in A:
```

```
        for col in linha:
```

```
            print("{:4}".format(col), end="")
```

```
        print()
```

Resumo (uma linha)

Detalhes (várias linhas)

Linha em branco

- **Módulos também podem ser documentados dessa maneira**

- **Módulos também podem ser documentados dessa maneira**
- **Basta iniciar o arquivo com o texto entre três aspas**

- `input()` e `print()` permitem interagir com o “mundo”

- `input()` e `print()` permitem interagir com o “mundo”
- Mas às vezes o “mundo” é um arquivo!

- `input()` e `print()` permitem interagir com o “mundo”
- Mas às vezes o “mundo” é um arquivo! **#comofaz?**

- `input()` e `print()` permitem interagir com o “mundo”
- Mas às vezes o “mundo” é um arquivo! **#comofaz?**
- `open()`, `read()`, `write()`, `close()`

```
arq = open("arquivo.txt", "r")  
print(arq.read())  
arq.close()
```

```
arq = open("arquivo.txt", "r")  
print(arq.read())  
arq.close()
```

```
arq = open("arquivo.txt", "r")  
linha = arq.readline()  
while linha != "":  
    ...  
    linha = arq.readline()  
arq.close()
```

```
arq = open("arquivo.txt", "r")
print(arq.read())
arq.close()
```

```
arq = open("arquivo.txt", "r")
linha = arq.readline()
while linha != "":
    ...
    linha = arq.readline()
arq.close()
```

```
arq = open("arquivo.txt", "r")
for linha in arq:
    ...
arq.close()
```

Arquivos

```
arq = open("arquivo.txt", "r")
print(arq.read())
arq.close()
```

```
arq = open("arquivo.txt", "r")
linha = arq.readline()
while linha != "":
    ...
    linha = arq.readline()
arq.close()
```

```
arq = open("arquivo.txt", "r")
for linha in arq:
    ...
arq.close()
```

```
arq = open("arquivo.txt", "r")
linhas = arq.readlines()
arq.close()
for linha in linhas:
    ...
```

```
poeminha = [ "Batatinha quando nasce",  
             "Espalha a rama pelo chão",  
             "A menina quando dorme",  
             "Põe a mão no coração" ]  
  
arq = open("arquivo.txt", "w")  
for linha in poeminha:  
    arq.write(linha + "\n")  
arq.close()
```

```
poeminha = [ "Batatinha quando nasce",  
             "Espalha a rama pelo chão",  
             "A menina quando dorme",  
             "Põe a mão no coração" ]  
  
arq = open("arquivo.txt", "w")  
for linha in poeminha:  
    arq.write(linha + "\n")  
arq.close()
```

```
poeminha = [ "Batatinha quando nasce",  
             "Espalha a rama pelo chão",  
             "A menina quando dorme",  
             "Põe a mão no coração" ]  
  
arq = open("arquivo.txt", "w")  
arq.write("\n".join(poeminha))  
arq.close()
```

Arquivos

```
poeminha = [ "Batatinha quando nasce",  
             "Espalha a rama pelo chão",  
             "A menina quando dorme",  
             "Põe a mão no coração" ]  
  
arq = open("arquivo.txt", "w")  
for linha in poeminha:  
    arq.write(linha + "\n")  
arq.close()
```

```
poeminha = [ "Batatinha quando nasce",  
             "Espalha a rama pelo chão",  
             "A menina quando dorme",  
             "Põe a mão no coração" ]  
  
arq = open("arquivo.txt", "w")  
arq.write("\n".join(poeminha))  
arq.close()
```

```
poeminha = "Batatinha quando nasce\n" \  
           "Espalha a rama pelo chão\n" \  
           "A menina quando dorme\n" \  
           "Põe a mão no coração"  
  
arq = open("arquivo.txt", "w")  
arq.write(poeminha)  
arq.close()
```


Arquivos

```
poeminha = [ "Batatinha quando nasce",  
             "Espalha a rama pelo chão",  
             "A menina quando dorme",  
             "Põe a mão no coração" ]  
  
arq = open("arquivo.txt", "w")  
for linha in poeminha:  
    arq.write(linha + "\n")  
arq.close()
```

```
poeminha = [ "Batatinha quando nasce",  
             "Espalha a rama pelo chão",  
             "A menina quando dorme",  
             "Põe a mão no coração" ]  
  
arq = open("arquivo.txt", "w")  
arq.write("\n".join(poeminha))  
arq.close()
```

```
poeminha = "Batatinha quando nasce\n" \  
           "Espalha a rama pelo chão\n" \  
           "A menina quando dorme\n" \  
           "Põe a mão no coração"  
  
arq = open("arquivo.txt", "w")  
arq.write(poeminha)  
arq.close()
```

```
poeminha = ("Batatinha quando nasce\n"  
           "Espalha a rama pelo chão\n"  
           "A menina quando dorme\n"  
           "Põe a mão no coração")  
  
arq = open("arquivo.txt", "w")  
for linha in poeminha.split("\n"):  
    arq.write(linha + "\n")  
arq.close()
```

Exercícios

Exercício – Numerando linhas

Escreva uma função que lê um arquivo e o imprime com as linhas numeradas.



Exercício – Numerando linhas

Escreva uma função que lê um arquivo e o imprime com as linhas numeradas.

```
def numera_linhas(nome):
```

Exercício – Numerando linhas

Escreva uma função que lê um arquivo e o imprime com as linhas numeradas.

```
def numera_linhas(nome):  
    f = open(nome, "r")
```

Exercício – Numerando linhas

Escreva uma função que lê um arquivo e o imprime com as linhas numeradas.

```
def numera_linhas(nome):  
    f = open(nome, "r")
```

```
    f.close()
```

Exercício – Numerando linhas

Escreva uma função que lê um arquivo e o imprime com as linhas numeradas.

```
def numera_linhas(nome):  
    f = open(nome, "r")  
    n = 1  
  
    f.close()
```

Exercício – Numerando linhas

Escreva uma função que lê um arquivo e o imprime com as linhas numeradas.

```
def numera_linhas(nome):  
    f = open(nome, "r")  
    n = 1  
  
    while True:  
        n += 1  
        f.close()
```


Exercício – Numerando linhas

Escreva uma função que lê um arquivo e o imprime com as linhas numeradas.

```
def numera_linhas(nome):  
    f = open(nome, "r")  
    n = 1  
    for linha in f:  
        n += 1  
    f.close()
```

Exercício – Numerando linhas

Escreva uma função que lê um arquivo e o imprime com as linhas numeradas.

```
def numera_linhas(nome):  
    f = open(nome, "r")  
    n = 1  
    for linha in f:  
        print("{}:".format(n), linha)  
        n += 1  
    f.close()
```

Exercício – Numerando linhas

Escreva uma função que lê um arquivo e o imprime com as linhas numeradas.

```
def numera_linhas(nome):  
    f = open(nome, "r")  
    n = 1  
    for linha in f:  
        print("{}:".format(n), linha)  
        n += 1  
    f.close()
```

Oops! Linhas “dobradas”!

Exercício – Numerando linhas

Escreva uma função que lê um arquivo e o imprime com as linhas numeradas.

```
def numera_linhas(nome):  
    f = open(nome, "r")  
    n = 1  
    for linha in f:  
        print("{}:".format(n), linha[:-1])  
        n += 1  
    f.close()
```

Exercício – Numerando linhas

Escreva uma função que lê um arquivo e o imprime com as linhas numeradas.

```
def numera_linhas(nome):  
    f = open(nome, "r")  
    n = 1  
    for linha in f:  
        print("{}:".format(n), linha[:-1])  
        n += 1  
    f.close()
```

E a última linha?

Exercício – Numerando linhas

Escreva uma função que lê um arquivo e o imprime com as linhas numeradas.

```
def numera_linhas(nome):  
    f = open(nome, "r")  
    n = 1  
    for linha in f:  
        print("{}:".format(n), linha, end="")  
        n += 1  
    f.close()
```

Exercício – Numerando linhas

Escreva uma função que lê um arquivo e o imprime com as linhas numeradas.

```
def numera_linhas(nome):  
    f = open(nome, "r")  
    n = 1  
    for linha in f:  
        print("{}:".format(n), linha, end="")  
        n += 1  
    f.close()
```

E a última linha?

Exercício – Numerando linhas

Escreva uma função que lê um arquivo e o imprime com as linhas numeradas.

```
def numera_linhas(nome):  
    f = open(nome, "r")  
    n = 1  
    for linha in f:  
        print("{}:".format(n), linha.rstrip())  
        n += 1  
    f.close()
```


Exercício – Numerando linhas

Escreva uma função que lê um arquivo e o imprime com as linhas numeradas.

```
def numera_linhas(nome):  
    f = open(nome, "r")  
    n = 1  
    for linha in f:  
        print("{}:".format(n), linha.rstrip())  
        n += 1  
    f.close()
```

E o alinhamento?

Exercício – Numerando linhas

Escreva uma função que lê um arquivo e o imprime com as linhas numeradas.

```
def numera_linhas(nome):  
    f = open(nome, "r")  
    n = 1  
    for linha in f:  
        print("{:4d}: {}".format(n, linha.rstrip()))  
        n += 1  
    f.close()
```

Arquivos – serialização 1/2

Dependendo da necessidade, pode valer a pena usar algo mais sofisticado:

- **Python pickle:** docs.python.org/3/library/pickle.html
 - ▶ Só funciona com python, muito simples de usar (quaisquer tipos de dados do python, mesmo os definidos pelo usuário, podem ser gravados e lidos sem conversões ou passos adicionais), arquivos maliciosos podem causar problemas
- **JSON:** docs.python.org/3/library/json.html
 - ▶ Excelente interoperabilidade; são arquivos de texto, então podem ser lidos e modificados com editores de texto comuns ou programas como o jq
- **YAML:** pyyaml.org
 - ▶ Similar ao anterior, mas mais poderoso (porém mais complexo)

Arquivos – serialização 2/2

Dependendo da necessidade, pode valer a pena usar algo mais sofisticado:

- **CSV:** docs.python.org/3/library/csv.html
 - ▶ Os dados são guardados em uma tabela em modo texto que pode ser aberta em planilhas eletrônicas como LibreOffice Calc, MS Excel etc. (essas planilhas também exportam para esse formato, então é possível ler dados gerados por elas com python)
- **SQLite:** docs.python.org/3/library/sqlite3.html
 - ▶ Permite fazer buscas e outras operações complexas com facilidade usando a linguagem SQL
- **Bibliotecas para formatos de arquivos específicos,** como pillow (python-pillow.org) para imagens, wave (docs.python.org/3/library/wave.html) para arquivos de áudio etc.

- **Escreva um programa que leia uma linha com palavras, separadas por espaços em branco, e determine o comprimento da maior palavra. Assuma que o texto não contém caracteres de pontuação.**