AULA 2

PI PARALELO

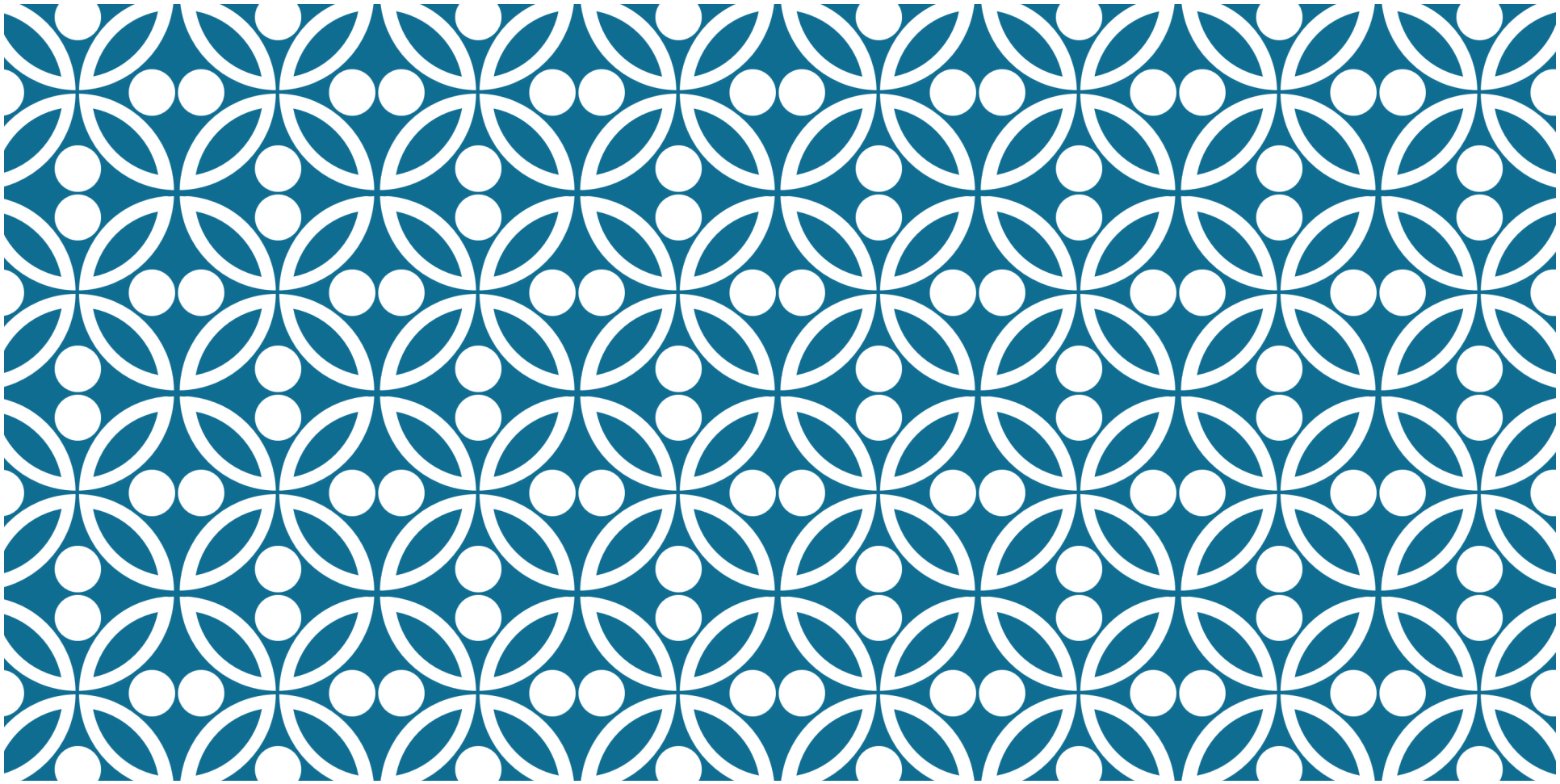
```
from multiprocessing import Process, Queue
import time
PROCS = 2
def pi(start, end, step):
    print "Start: " + str(start)
    print "End: " + str(end)
    sum = 0.0
    for i in range(start, end):
        x = (i+0.5) * step
        sum = sum + 4.0/(1.0+x*x)
    print(sum)
if __name__ == "__main__":
    num_steps = 100000000 #100.000.000
    sum = 0.0
    step = 1.0/num_steps
    proc_size = num_steps / PROCS

    tic = time.time()
    workers = []
    for i in range(PROCS):
        worker = Process(target=pi, args=(i*proc_size, (i+1)*proc_size - 1, step, ))
        workers.append(worker)
    for worker in workers :
        worker.start()
    for worker in workers :
        worker.join()
    toc = time.time()

    print "Pi: %.8f s" %(toc-tic)
```

PI PARALELO

```
from multiprocessing import Process, Queue
import time
PROCS = 2
def pi(start, end, step):
    print "Start: " + str(start)
    print "End: " + str(end)
    sum = 0.0
    for i in range(start, end):
        x = (i+0.5) * step
        sum = sum + 4.0/(1.0+x*x)
    print(sum)
if __name__ ==
    num_steps
    sum = 0.0
    step = 1.0
    proc_size

    tic = time.time()
    workers = []
    for i in range(PROCS):
        worker = Process(target=pi, args=(i*proc_size, (i+1)*proc_size - 1, step, ))
        workers.append(worker)
```

```
    for worker in workers :
        worker.start()
    for worker in workers :
        worker.join()
    toc = time.time()

    print "Pi: %.8f s" %(toc-tic)
```

Fácil, agora é só somar os resultados da saída. Depois multiplicar por step.



TROCANDO DADOS ENTRE PROCESSOS USANDO **QUEUE**

EXEMPLO 1
PROCESSOS NÃO COMPARTILHAM MEMÓRIA!

```
from multiprocessing import Process, current_process
import itertools
ITEMS = [1, 2, 3, 4, 5, 6]

def worker(items) :
  for i in itertools.count():
    try :
      items.pop()
    except IndexError :
      break
  print current_process().name, "processed %i items . " %i

if __name__ == "__main__" :
  workers = [ Process (target = worker, args = (ITEMS, ) ) for i in range (3) ]
  for worker in workers :
    worker.start()
  for worker in workers :
    worker.join()
  print "ITEMS after all workers finished : " , ITEMS
```

Saída:
Process 1 processed 6 items .
Process 2 processed 6 items .
Process 3 processed 6 items .
ITEMS after all workers finished : [1, 2, 3, 4, 5, 6]

INTER PROCESS COMMUNICATION (IPC) USANDO QUEUES

Queue (Fila)

- Multi-consumidor, multi-consumidor FIFO
- Vários processos podem colocar itens na Queue, outros podem obtê-los
- Queues são thread e process safe.

Atenção: Se um processo for eliminado usando Process.terminate() ou os.kill() enquanto estiver tentando usar uma Fila, os dados da fila provavelmente ficarão corrompidos. Isso pode fazer com que qualquer outro processo receba uma exceção quando tentar usar a fila mais tarde.

INTER PROCESS COMMUNICATION (IPC) USANDO QUEUES

put(obj[, block[, timeout]])

- Coloque obj na fila.
- Se argumento opcional Block for True (padrão) e o tempo limite for None (padrão), bloqueie se necessário até que um slot livre esteja disponível.
- Se o tempo limite for um número positivo, ele bloqueará no máximo dois segundos do tempo limite e gerará a exceção queue.Full se nenhum slot livre estiver disponível dentro desse tempo.
- Caso contrário (block for False), coloque um item na fila se um slot livre estiver imediatamente disponível, caso contrário, gere a exceção queue.Full (o tempo limite é ignorado nesse caso).

get([block[, timeout]])

- Remova e retorne um item da fila.
- Se o bloco opcional de argumentos for True (padrão) e o tempo limite for None (padrão), bloqueie, se necessário, até que um item esteja disponível.
- Se o tempo limite for um número positivo, ele bloqueará no máximo dois segundos do tempo limite e gerará a exceção queue.Empty se nenhum item estiver disponível dentro desse tempo.
- Caso contrário (block for False), retorne um item se um estiver imediatamente disponível, senão, gere a exceção queue.Empty (o tempo limite é ignorado nesse caso).

INTER PROCESS COMMUNICATION (IPC) USANDO QUEUES

qsize()

- Retornar o tamanho aproximado da fila.
- Por causa da semântica multithreading / multiprocessamento, esse número não é confiável.
- Note que isso pode gerar a exceção NotImplementedError em plataformas Unix como o Mac OS X, onde sem_getvalue() não está implementado.

empty()

- Retorne True se a fila estiver vazia, caso contrário, False.
- Por causa da semântica multithreading / multiprocessamento, isso não é confiável.

full()

- Retorne True se a fila estiver cheia, caso contrário, False.
- Por causa da semântica multithreading / multiprocessamento, isso não é confiável.

EXEMPLO 2
USANDO MULTIPROCESSAMENTO.QUEUE

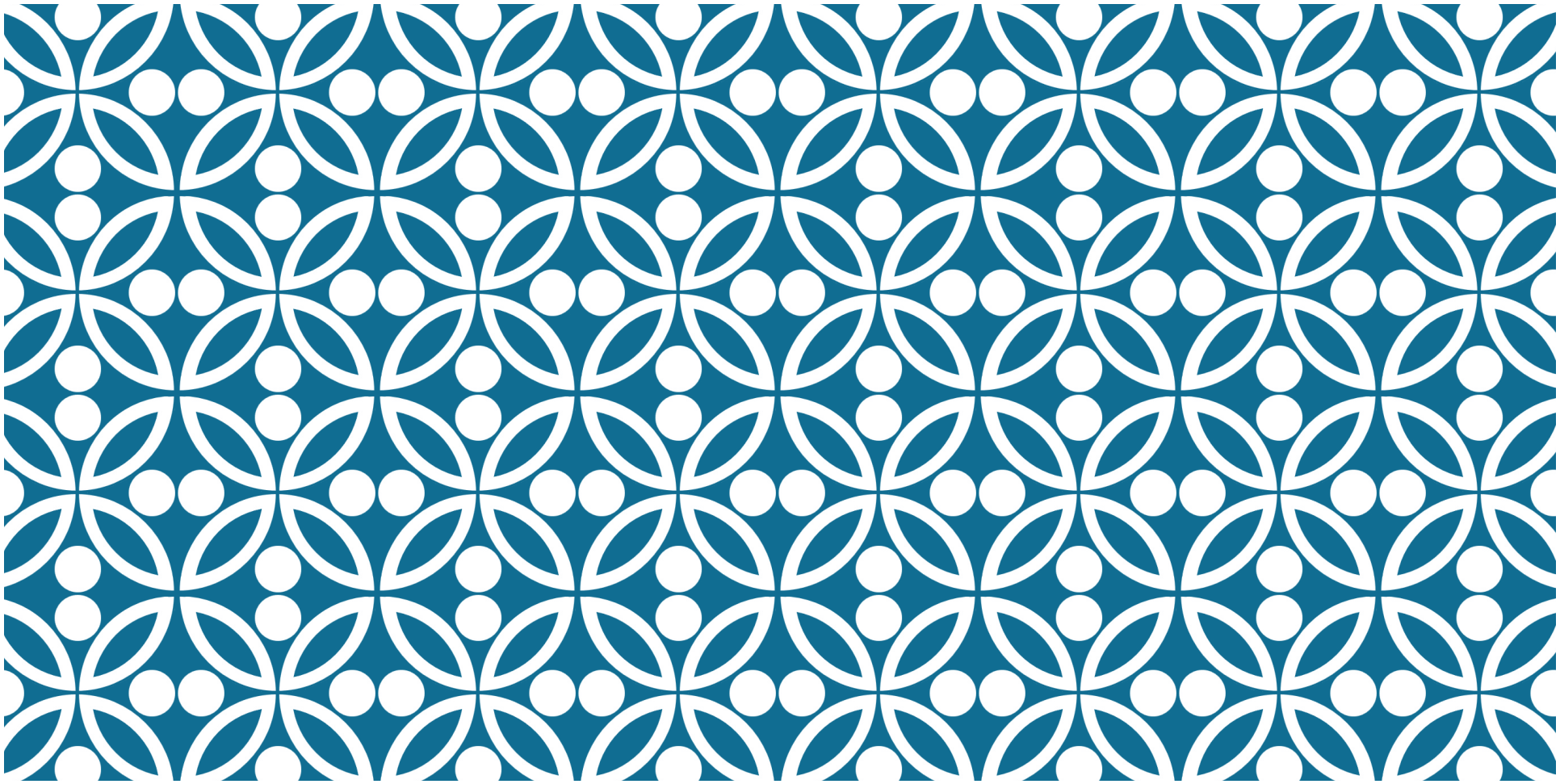
```
from multiprocessing import Process, current_process, Queue
import itertools

ITEMS = Queue()
for i in [1, 2, 3, 4, 5, 6, 'end' , 'end' , 'end' ] :
  ITEMS.put(i)

def worker(items) :
  for i in itertools.count() :
    item = items.get()
    if item == 'end':
      break
  print current_process().name, "processed %i items . " %i

if __name__ == "__main__" :
  workers = [ Process ( target=worker , args =(ITEMS, ) ) for i in range(3) ]
  for worker in workers :
    worker.start( )
  for worker in workers :
    worker.join( )
print "#ITEMS after all workers finished : ", ITEMS.qsize( )
```

Saída:

Process 1 processed 1 items .
Process 2 processed 5 items .
Process 3 processed 0 items .
#ITEMS after all workers finished : 0

EXERCÍCIO

Transforme o programa Pi em paralelo + QUEUE !



EXCEÇÕES

EXCEÇÕES

Mesmo se uma instrução ou expressão estiver sintaticamente correta, isso poderá causar um erro quando for feita uma tentativa de executá-la.

Os erros detectados durante a execução são chamados de exceções e não são incondicionalmente fatais: em breve você aprenderá como lidar com eles em programas Python.

A maioria das exceções não são tratadas pelos programas, no entanto, e resultam em mensagens de erro, conforme mostrado aqui:

EXCEÇÕES

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

EXCEÇÕES

A última linha da mensagem de erro indica o que aconteceu.

Exceções vêm em diferentes tipos e o tipo é impresso como parte da mensagem: os tipos no exemplo são ZeroDivisionError, NameError e TypeError.

A sequência impressa como o tipo de exceção é o nome da exceção interna que ocorreu.

Isso é verdadeiro para todas as exceções internas, mas não precisa ser verdadeiro para exceções definidas pelo usuário (embora seja uma convenção útil).

Nomes de exceção padrão são identificadores internos (não são palavras-chave reservadas).

EXCEÇÕES

É possível escrever programas que manipulam exceções selecionadas. Veja o exemplo a seguir

```
>>> while True:
...     try:
...         x = int(input("Please enter a number: "))
...         break
...     except ValueError:
...         print("Oops!  That was no valid number.  Try again...")
...
```

EXCEÇÕES

A instrução try funciona da seguinte maneira:

Primeiro, a cláusula try (a declaração entre as palavras-chave try e except) é executada.

Se nenhuma exceção ocorrer, a cláusula except será ignorada e a execução da instrução try será concluída.

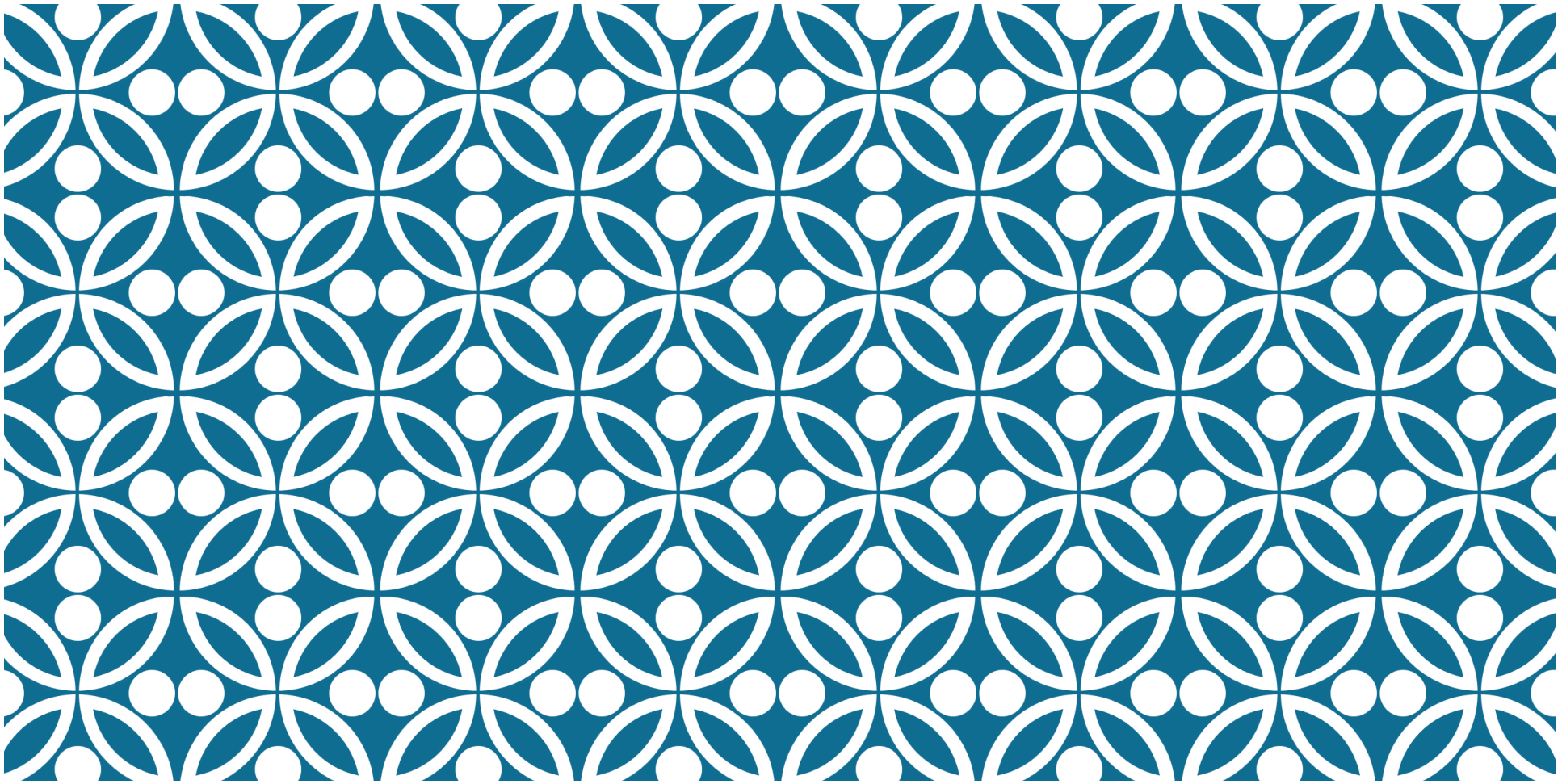
Se ocorrer uma exceção durante a execução da cláusula try, o resto da cláusula será ignorada. Então, se seu tipo corresponder à exceção nomeada após a palavra-chave except, a cláusula except será executada e a execução continuará após a instrução try.

Se ocorrer uma exceção que não corresponda à exceção nomeada na cláusula except, ela será passada para as instruções try externas; se nenhum manipulador for encontrado, é uma exceção não tratada e a execução é interrompida

TRATANDO EXCEÇÕES

```
def divide(x, y):
  try:
    result = x / y
  except ZeroDivisionError:
    print("division by zero!")
  else:
    print("result is", result)
```

O uso da cláusula else é melhor do que adicionar código adicional à cláusula try, pois evita a captura acidental de uma exceção que não foi gerada pelo código protegido pela instrução try... except.



TROCANDO DADOS ENTRE PROCESSOS USANDO **PIPES**

INTER PROCESS COMMUNICATION (IPC) USANDO PIPES

Pipe (Tubo)

- Para comunicação entre dois processos
- Um Pipe tem duas extremidades: o processo A escreve algo em sua extremidade do Pipe e o processo B pode lê-lo de sua
- Pipes são bidirecionais

Atenção: Os dados de um pipe podem ficar corrompidos se dois processos (ou threads) tentarem ler ou gravar na mesma extremidade do pipe ao mesmo tempo.

Naturalmente, não há risco de corrupção de processos usando diferentes extremidades do tubo ao mesmo tempo.

Atenção: Se um processo for eliminado enquanto estiver tentando ler ou gravar em um pipe, é provável que os dados no pipe se tornem corrompidos, porque pode ser impossível ter certeza de onde estão os limites de mensagens.

PIPES

Um pipe tem duas extremidades: a, b = Pipe()

Um processo envia algo para um lado e o outro processo pode receber do outro

recv irá bloquear se o pipe estiver vazio

Fato interessante

- As filas são implementadas usando Pipes + bloqueios.

INTER PROCESS COMMUNICATION (IPC) USANDO PIPES

send(obj)

- Envie um objeto para a outra extremidade da conexão, que deve ser lida usando recv ().
- O objeto deve ser particionável. Partições muito grandes (aproximadamente +32 MB, embora dependa do sistema operacional) podem gerar uma exceção ValueError.

recv()

- Retorna um objeto enviado da outra extremidade da conexão usando send().
- Bloqueia até que haja algo para receber.
- Gera a exceção EOFError se não houver mais nada para receber e o outro lado estiver fechado.

INTER PROCESS COMMUNICATION (IPC) USANDO PIPES

send_bytes(buffer[, offset[, size]])

- Envie dados de byte de um objeto semelhante a bytes como uma mensagem completa.
- Se offset é dado, então os dados são lidos daquela posição no buffer.
- Se o tamanho for informado, então muitos bytes serão lidos do buffer.
- Buffers muito grandes (aproximadamente +32 MB, embora dependa do sistema operacional) podem gerar uma exceção ValueError

recv_bytes([maxlength])

- Retorna uma mensagem completa de dados de byte enviados da outra extremidade da conexão como uma string.
- Bloqueia até que haja algo para receber.
- Gera a exceção EOFError se não houver mais nada para receber e o outro lado estiver fechado.
- Se maxlength for especificado e a mensagem for maior que maxlength, a exceção OSError será gerada e a conexão não será mais legível.

EXEMPLO 3: COMUNICADOR PIPE

```
from multiprocessing import Process , Pipe
def worker(conn) :
  while True :
    item = conn.recv()
    if item == 'end' :
      break
    print item
def master(conn) :
  conn.send(' Is')
  conn.send(' this')
  conn.send(' on?')
  conn.send('end')
if __name__ == '__main__' :
  a, b = Pipe()
  w = Process(target = worker, args = (a, ) )
  m = Process(target = master, args = (b, ) )
  w.start()
  m.start()
  w.join()
  m.join()
```

Saída:
Is
this
on?

Será que o trabalhador pode falar com o mestre?

EXEMPLO 3: COMUNICADOR PIPE

```
from multiprocessing import Process , Pipe
def worker(conn) :
  while True :
    item = conn.recv()
    if item == 'end' :
      break
    print item
  conn.send('thanks')

def master(conn) :
  conn.send(' Is')
  conn.send(' this')
  conn.send(' on?')
  conn.send('end')
  item = conn.recv()
  print item
```

```
if __name__ == '__main__' :
  a, b = Pipe()
  w = Process(target = worker, args = (a, ) )
  m = Process(target = master, args = (b, ) )
  w.start()
  m.start()
  w.join()
  m.join()
```

Saída:
Is
this
on?
Thanks

EXERCÍCIO

Transforme o programa Pi em paralelo + PIPES!

EXERCÍCIO

Transforme o programa Pi em paralelo + PIPES!

Atenção: Os dados de um pipe podem ficar corrompidos se dois processos (ou threads) tentarem ler ou gravar na mesma extremidade do pipe ao mesmo tempo.