Python em Uma Linha

https://github.com/JoaoFelipe/presentations/

João Felipe N. Pimentel

joaofelipenp@gmail.com

Instituto de Computação

Laboratório de Engenharia de Sistemas e Informação

Índice

Índice

- Como assim em uma linha?

Como assim em uma linha?

```
In [1]: ((lambda source, target: (lambda itself: lambda target, visited, fif
```

```
Out[1]: 6
```

```
In [2]: ';' in In[1]
```

```
Out[2]: False
```

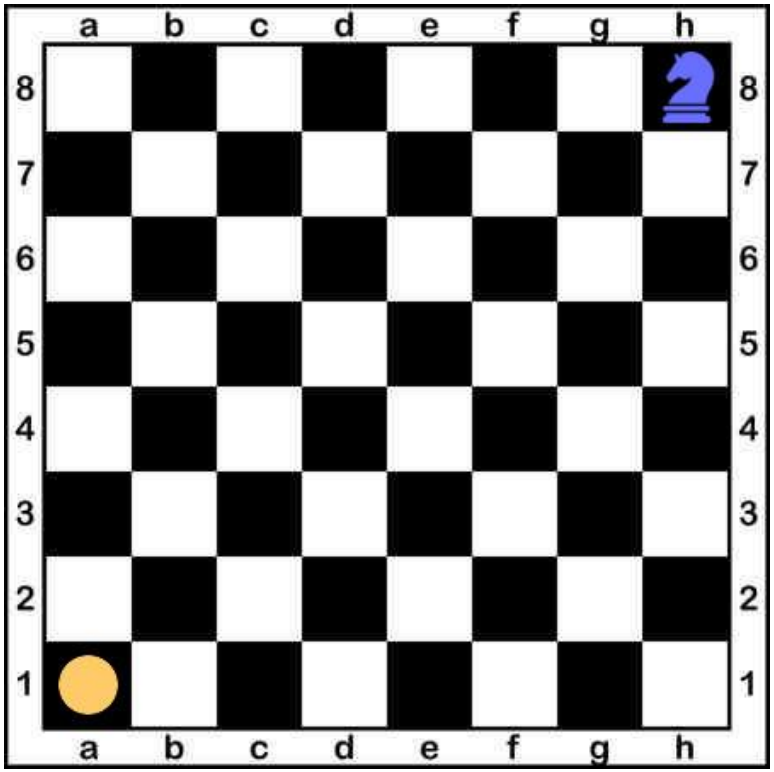
Como assim em uma linha?

In [1]:
```
((lambda source, target: (lambda itself: lambda target, visited,
fifo: itself(target, visited, fifo, itself))(lambda target,
visited, fifo, itself: ((lambda chess_to_tuple, tuple_to_chess:
((lambda knight_moves: (itself(target, *(lambda element, *rest:
(lambda filtered: ({**visited, **{pos: visited[element] + 1 for pos
in filtered}},list(rest) + filtered if target not in filtered else
[target]))([x for x in knight_moves(element) if x not in
visited]))(*fifo), itself) if fifo else visited[target]))(lambda
pos: (lambda tup: filter(lambda y: y is not None,map(lambda x:
tuple_to_chess(x[0] + tup[0], x[1] + tup[1]),[(1, -2), (-1, 2), (-
2, -1), (1, 2),(2, -1), (-2, 1), (-1, -2), (2,
1)])))(chess_to_tuple(pos)))))(lambda pos: (ord(pos[0]) - ord('a')
+ 1, int(pos[1])),lambda col, lin: chr(col + ord('a') - 1) +
str(lin) if 1 <= col <= 8 and 1 <= lin <= 8 else None)))(target,
{source: 0}, [source])))("h8", "a1")
```

Out[1]: 6

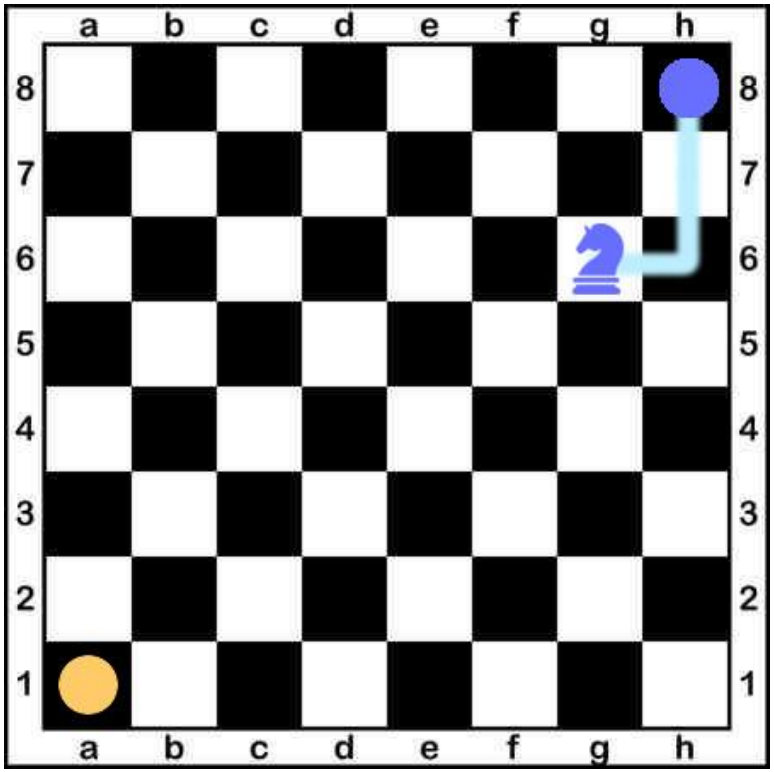
O que faz

- Quantos movimentos o cavalo do xadrez precisa para ir da posição h8 até a1



O que faz

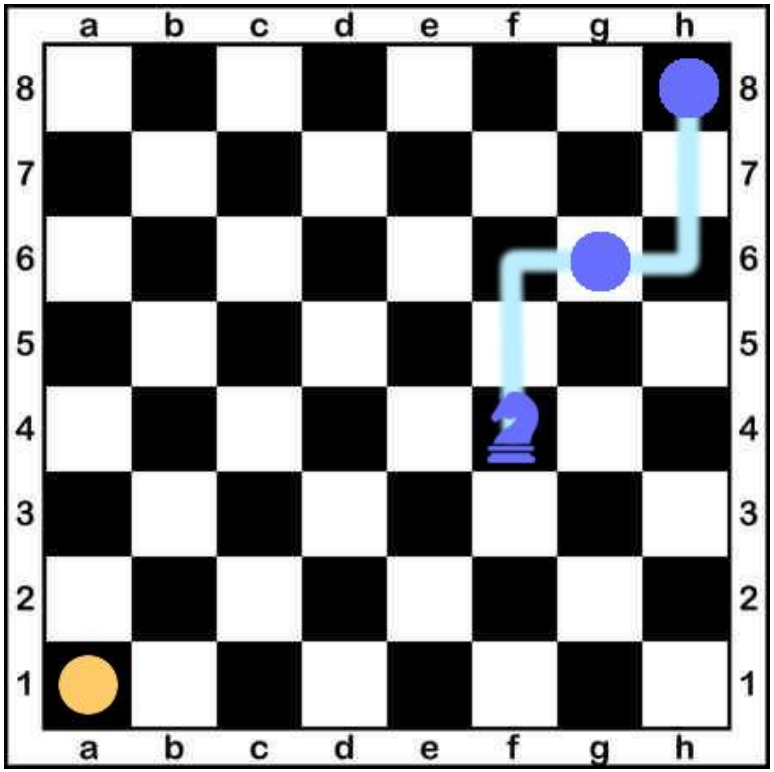
- Quantos movimentos o cavalo do xadrez precisa para ir da posição h8 até a1



1

O que faz

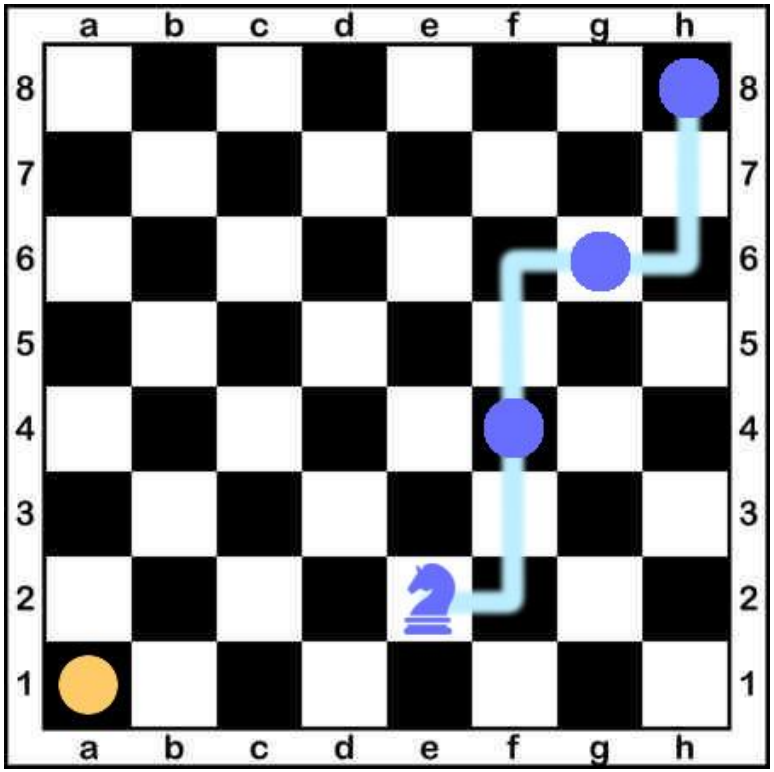
- Quantos movimentos o cavalo do xadrez precisa para ir da posição h8 até a1



2

O que faz

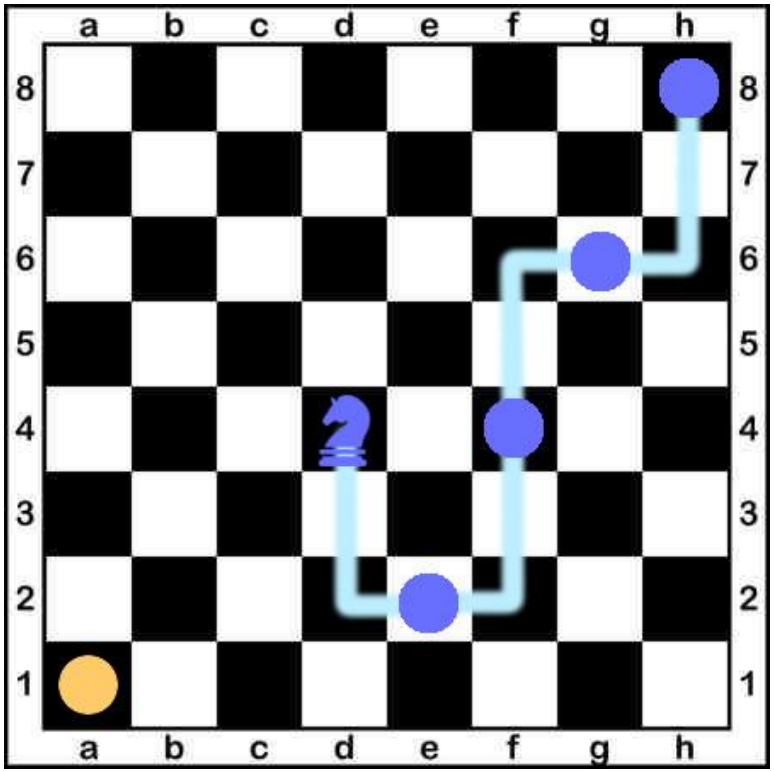
- Quantos movimentos o cavalo do xadrez precisa para ir da posição h8 até a1



3

O que faz

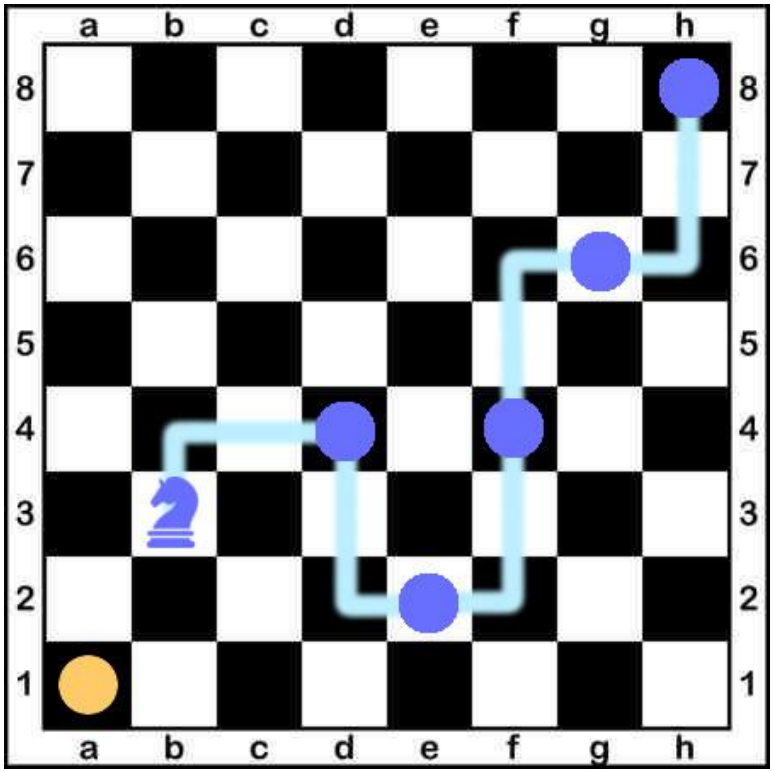
- Quantos movimentos o cavalo do xadrez precisa para ir da posição h8 até a1



4

O que faz

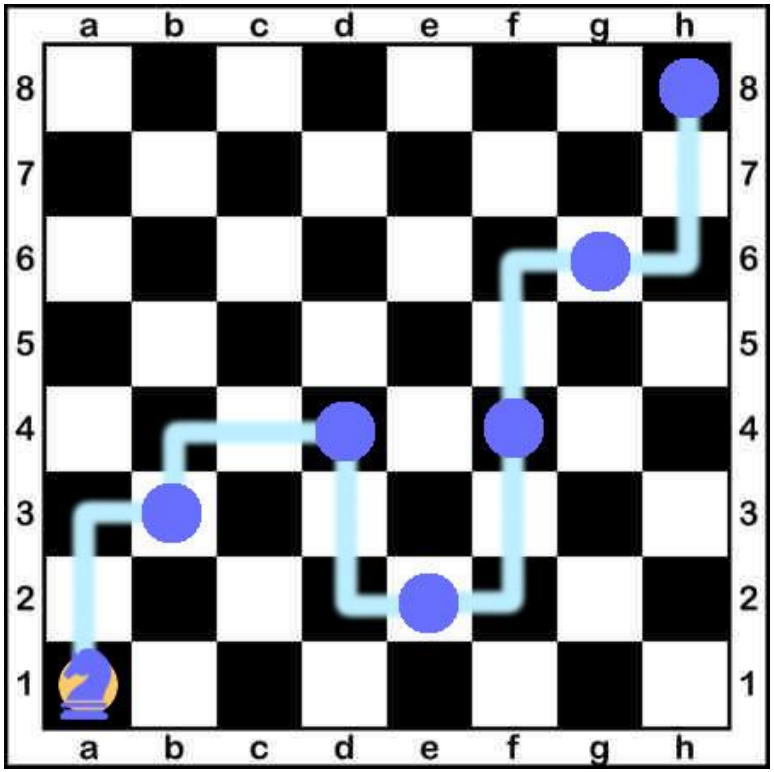
- Quantos movimentos o cavalo do xadrez precisa para ir da posição h8 até a1



5

O que faz

- Quantos movimentos o cavalo do xadrez precisa para ir da posição h8 até a1



6

Código equivalente (1/4)

In [1]:

```
from collections import deque

def knight(source, target):
    "Calculates the minimum number of steps"
    visited = {source: 0}
    fifo = deque([source])
    if source == target:
        return 0
    while fifo:
        element = fifo.popleft()
        for pos in knight_moves(element):
            if not pos in visited:
                visited[pos] = visited[element] + 1
                fifo.append(pos)
            if pos == target:
                return visited[pos]
```

Código equivalente (2/4)

In [2]:

```
def knight_moves(chess_position):
    "Generates possible movement positions for the knight"
    position_tuple = chess_to_tuple(chess_position)
    delta_moves = [
        (1, -2), (-1, 2), (-2, -1), (1, 2),
        (2, -1), (-2, 1), (-1, -2), (2, 1)
    ]
    for delta in delta_moves:
        new_position = tuple_to_chess(
            position_tuple[0] + delta[0],
            position_tuple[1] + delta[1]
        )
        if new_position is not None:
            yield new_position
```

Código equivalente (3/4)

In [3]:

```
def chess_to_tuple(chess_position):
    "Converts chess position (eg., 'a1') to tuple (eg., (1, 1))"
    first = ord(chess_position[0]) - ord('a') + 1
    second = int(chess_position[1])
    return (first, second)


def tuple_to_chess(column, line):
    "Converts position to chess notation (eg., 'a1')"
    if 1 <= column <= 8 and 1 <= line <= 8:
        first = chr(column + ord('a') - 1)
        second = str(line)
        return first + second
    return None
```

Código equivalente (4/4)

```
In [4]: knight("h8", "a1")
```

```
Out[4]: 6
```

Índice

Por quê?

Por quê?

- ASCII Art – Código Equivalente
  - Não precisa respeitar identações do Python

In [1]:
```
knight = (
                                                  lambda
                                              source,   target: (
                                          lambda           itself:
                                      lambda               target,
                                  visited,                      fifo:
                              itself                              (target,
                          visited,           fifo,                 itself))(
                  lambda target             ,visited,              fifo, itself: (
              (lambda                    chess_to_tuple,            tuple_to_chess: (
            (lambda                                                   knight_moves: (
          itself(                                                           target,
     *(lambda                                                               element,
     *rest:       (lambda                                                filtered: (
{**                visited,                                                  **{pos:
visited                                                                    [element]
+ 1 for pos                                                                in filtered
      }}, list                                                               (rest) +
   filtered                                                                 if target
      not in filtered                                                            else [
        target]))([x                                                      for x in
           knight_moves(element) if x not                                in visited]
                                  ))(*fifo),                          itself) if fifo
                              else                                          visited
                           [target]))(                                       lambda
                        pos:                                                 (lambda
                     tup:                                                     filter
                  (lambda                                                      y: y
                 is not                                                       None,
                 map(                                                     lambda x:
              tuple_to_chess (x[0] + tup[0], x[1] + tup[1]), [(1, -2), (-1, 2), (-2, -1),
               (1, 2),                                                      (2, -1),
              (-2, 1),                                                       (-1, -2),
              (2, 1)])))                                               (chess_to_tuple
             (pos)))                                                     ))(lambda pos:
             (ord                                                              (pos[0])
            - ord('a')                                                       + 1, int
             (pos[1])),                                                   lambda col,
             lin:                                                         chr(col +
             ord('a')                                                     - 1) + str
  (lin)  if  1  <=  col  <=  8  and  1  <=  lin  <=  8 else  None)))(target,   {source:  0},   [source]))

knight("h8", "a1")
```

Out[1]: 6

Por quê?

- ~~Job Security~~
  - Alguém na empresa entende isso?
  - Cuidado com programadores psicopatas que sabem onde você mora!

```
In [1]: knight = (
            lambda
          source,   target: (
        lambda          itself:
      lambda            target,
    visited,                fifo:
  itself                     (target,
 visited,          fifo,      itself))(
lambda target      ,visited,   fifo, itself: (
  (lambda          chess_to_tuple,   tuple_to_chess: (
    (lambda                            knight_moves: (
   itself(                                    target,
 *(lambda                                     element,
 *rest:       (lambda                       filtered: (
{**             visited,                       **{pos:
visited                                        [element]
+ 1 for pos                                    in filtered
    }}, list                                    (rest) +
 filtered                                      if target
   not in filtered                                  else [
    target]))([x                               for x in
      knight_moves(element) if x not          in visited]
                          ))(*fifo),         itself) if fifo
                      else                          visited
                    [target]))(                      lambda
                  pos:                               (lambda
                tup:                                 filter
              (lambda                                  y: y
            is not                                    None,
            map(                                   lambda x:
          tuple_to_chess (x[0] + tup[0], x[1] + tup[1]), [(1, -2), (-1, 2), (-2, -1),
         (1, 2),                                          (2, -1),
        (-2, 1),                                          (-1, -2),
        (2, 1)]))                                  (chess_to_tuple
       (pos)))                                      ))(lambda pos:
       (ord                                               (pos[0])
     - ord('a')                                             + 1, int
      (pos[1])),                                          lambda col,
      lin:                                                chr(col +
      ord('a')                                            -1) + str
 (lin)  if  1  <=  col  <=  8  and  1  <=  lin  <=  8 else  None)))(target,  {source:  0},  [source]))

knight("h8", "a1")

Out[1]: 6
```

Por quê?

- Motivo Real: Aprender Python Intermediário
  - Slice
  - If Expression
  - Funções anônimas (lambda)
  - Decorator
  - functools
  - List/Set/Dict Comprehensions
  - Geradores/Gen Expressions
  - itertools

Índice

Comandos vs Expressões

- Python usa identação para identificar escopos
  - Comandos dificultar implementar em uma linha

Comandos

- Inserção de elemento no meio de uma lista
- 3 comandos: 1 atribuição e 2 chamadas de função
  - Chamadas são expressões. Um comando pode ser uma expressão. Stmt ::= Expr

In [1]:
```
a = [7, 9, 10, 11, 12, 13, 14, 15]
a.insert(1, 8)
print(a)
```

Out[1]: [7, 8, 9, 10, 11, 12, 13, 14, 15]

Comandos

- Em uma linha
  - Usando ; para sequências de comandos:

```
In [1]: a = [7, 9, 10, 11, 12, 13, 14, 15]; a.insert(1, 8); print(a)

Out[1]: [7, 8, 9, 10, 11, 12, 13, 14, 15]
```

Comandos

- Não funciona sempre:

In [3]:
```
if not a:
    print("Lista vazia")
print("Fora do if")
```

```
Fora do if
```

- Usando ; não dá pra saber o escopo:

In [3]:
```
if not a: print("Lista vazia"); print("Fora do if")
```

```
<vazio>
```

```
a == [7, 8, 9, 10, 11, 12, 13, 14, 15]
```

Expressão

- Inserção de elemento no meio de uma lista
  - Usando *slice* e concatenação:

```
In [2]: a[:1] + [8] + a[1:]
```

```
Out[2]: [7, 8, 9, 10, 11, 12, 13, 14, 15]
```

```
a == [7, 9, 10, 11, 12, 13, 14, 15]
```

Slice

- Um “pedaço” de uma lista ou tupla:

```
list[inicio:fim:passo]
```

- Fechado em *início*, aberto em *fim*, iterando de *passo* em *passo*
- Qualquer termo pode ser omitido. Por padrão:

```
inicio = 0
fim = len(lista)
passo = 1
```

Slice

- No exemplo (`a[:1] + [8] + a[1:]`):

```
a[:1] == [7]
a[1:] == [9, 10, 11, 12, 13, 14, 15]
```

- Outros slices:

```
a[2:5] == [10, 11, 12]
a[5:2:-1] == [13, 12, 11]
a[::2] == [7, 10, 12, 14]
```

```
a == [7, 9, 10, 11, 12, 13, 14, 15]
```

Expressão

- Transformando *if*
  - Anterior:

In [3]:
```
if not a:
    print("Lista vazia")
print("Fora do if")
```

```
Fora do if
```

  - Usando *if expression:*

In [3]:
```
print(("Lista vazia\n" if not a else "") + "Fora do if")
```

```
Fora do if
```

```
a == [7, 9, 10, 11, 12, 13, 14, 15]
```

If Expression

- Ternário

```
v if cond else f
```

- Retorna *v* se *cond* for verdadeiro
- Caso contrário, retorna *f*
- No exemplo:

```
"Lista vazia\n" if not a else ""
```

- Como a lista possui elementos, retorna “”

```
a == [7, 9, 10, 11, 12, 13, 14, 15]
```

Exercícios 1 - 3

1. Use *slice* para remover o último elemento de uma lista
2. Use *slice* para fazer uma cópia de uma lista
3. Use if expressions para reescrever:

```
x = 5
if x < 5:
    y = -1
elif x == 5:
    y = 0
else:
    y = 1
```

Índice

Funções

Instituto de Computação
ESI
Laboratório de Engenharia de Sistemas e Informação

- Função que calcula o próximo número da conjectura de collatz:

In [1]:
```
def next_collatz(x):
    """Next collatz conjecture number"""
    if x % 2 == 0:
        return x // 2
    else:
        return 3 * x + 1
```

In [2]:
```
next_collatz(5)
```

Out[2]: 16

In [3]:
```
next_collatz(16)
```

Out[3]: 8

In [4]:
```
next_collatz(8)
```

Out[4]: 4

Funções

- Usando *if expression*:

In [1]:
```
def next_collatz(x):
    return x // 2 if x % 2 == 0 else 3 * x + 1
```

In [2]:
```
next_collatz(5)
```

Out[2]: 16

In [3]:
```
next_collatz(16)
```

Out[3]: 8

In [4]:
```
next_collatz(8)
```

Out[4]: 4

Funções

- Usando *lambda expression*:

In [1]:
```
def next_collatz(x):
    return x // 2 if x % 2 == 0 else 3 * x + 1
```

In [1]:
```
next_collatz = lambda x: x // 2 if x % 2 == 0 else 3 * x + 1
```

In [2]:
```
next_collatz(5)
```

Out[2]: 16

In [3]:
```
next_collatz(16)
```

Out[3]: 8

In [4]:
```
next_collatz(8)
```

Out[4]: 4

Lambda Expression

- Função anônima

```
lambda <args>: <result>
```

- Função sem nome que recebe os argumentos *args* e retorna *result*
  - Result deve ser uma expressão
- No exemplo:

```
lambda x: x // 2 if x % 2 == 0 else 3 * x + 1
```

- A função resultante recebe *x* como parâmetro e retorna o resultado do ternário

Situação Comum: Ordenação

In [1]:
```
from collections import namedtuple

Score = namedtuple("Score", "country gold silver bronze")
scores = [
    Score("China", 26, 18, 26),
    Score("Brasil", 7, 6, 6),
    Score("Hungria", 8, 3, 4),
    Score("Canadá", 4, 3, 15),
    Score("Japão", 12, 8, 21),
]
```

- Ordenar:
  - Nome
  - Total de medalhas (soma)
  - Tipo de medalha (ouro > prata > bronze)

Situação Comum: Ordenação

In [1]:

```
from collections import namedtuple

Score = namedtuple("Score", "country gold silver bronze")
scores = [
    Score("China", 26, 18, 26),
    Score("Brasil", 7, 6, 6),
    Score("Hungria", 8, 3, 4),
    Score("Canadá", 4, 3, 15),
    Score("Japão", 12, 8, 21),
]
```

Tupla com estrutura definida

- Ordenar:
  - Nome
  - Total de medalhas (soma)
  - Tipo de medalha (ouro > prata > bronze)

Ordenando por nome

```
In [2]: sorted(scores)
```

```
Out[2]: [Score(country='Brasil', gold=7, silver=6, bronze=6),
         Score(country='Canadá', gold=4, silver=3, bronze=15),
         Score(country='China', gold=26, silver=18, bronze=26),
         Score(country='Hungria', gold=8, silver=3, bronze=4),
         Score(country='Japão', gold=12, silver=8, bronze=21)]
```

Ordenando por total de medalhas

```
In [3]: sorted(scores, reverse=True, key=lambda score: sum(score[1:]))
```

```
Out[3]: [Score(country='China', gold=26, silver=18, bronze=26),
         Score(country='Japão', gold=12, silver=8, bronze=21),
         Score(country='Canadá', gold=4, silver=3, bronze=15),
         Score(country='Brasil', gold=7, silver=6, bronze=6),
         Score(country='Hungria', gold=8, silver=3, bronze=4)]
```

- Ordem decrescente
- sum(score[1:]) é o mesmo que:

  score.gold + score.silver + score.bronze

  score[1] + score[2] + score[3]

Ordenando por tipo das medalhas (1/5)

In [4]:

```
from functools import cmp_to_key
@cmp_to_key
def score_cmp(first, second):
    if first.gold > second.gold: return 1
    elif first.gold < second.gold: return -1
    elif first.silver > second.silver: return 1
    elif first.silver < second.silver: return -1
    elif first.bronze > second.bronze: return 1
    elif first.bronze < second.bronze: return -1
    elif sum(first[1:]) > sum(second[1:]): return 1
    elif sum(first[1:]) < sum(second[1:]): return -1
    elif first.country > second.country: return 1
    elif first.country < second.country: return -1
    else: return 0
```

Ordenando por tipo das medalhas (2/5)

```
In [5]: sorted(scores, reverse=True, key=score_cmp)
```

```
Out[5]: [Score(country='China', gold=26, silver=18, bronze=26),
         Score(country='Japão', gold=12, silver=8, bronze=21),
         Score(country='Hungria', gold=8, silver=3, bronze=4),
         Score(country='Brasil', gold=7, silver=6, bronze=6),
         Score(country='Canadá', gold=4, silver=3, bronze=15)]
```

Ordenando por tipo das medalhas (3/5)

In [4]:
```
from functools import cmp_to_key
@cmp_to_key
def score_cmp(first, second):
    first_tuple = (first.gold, first.silver, first.bronze,
                   sum(first[1:]), first.country)
    second_tuple = (second.gold, second.silver, second.bronze,
                    sum(second[1:]), second.country)
    if first_tuple > second_tuple:
        return 1
    elif first_tuple < second_tuple:
        return -1
    else:
        return 0
```

In [5]:
```
sorted(scores, reverse=True, key=score_cmp)
```

Out[5]: ...

Ordenando por tipo das medalhas (4/5)

In [4]:
```
def score_key(score):
    return (score.gold, score.silver, score.bronze,
            sum(score[1:]), score.country)

sorted(scores, reverse=True, key=score_key)
```

Out[4]:
```
[Score(country='China', gold=26, silver=18, bronze=26),
 Score(country='Japão', gold=12, silver=8, bronze=21),
 Score(country='Hungria', gold=8, silver=3, bronze=4),
 Score(country='Brasil', gold=7, silver=6, bronze=6),
 Score(country='Canadá', gold=4, silver=3, bronze=15)]
```

- Deixamos de usar *cmp_to_key*
- Passamos a usar uma função key

Ordenando por tipo das medalhas (5/5)

```
In [4]: sorted(scores, reverse=True, key=lambda score: (score.gold,
        score.silver, score.bronze, sum(score[1:]), score.country))
```

```
Out[4]: [Score(country='China', gold=26, silver=18, bronze=26),
         Score(country='Japão', gold=12, silver=8, bronze=21),
         Score(country='Hungria', gold=8, silver=3, bronze=4),
         Score(country='Brasil', gold=7, silver=6, bronze=6),
         Score(country='Canadá', gold=4, silver=3, bronze=15)]
```

- Em uma linha

Usando lambda para substituir atribuições

- Score = ...

In [1]:
```
from collections import namedtuple
Score = namedtuple("Score", "country gold silver bronze")
scores = [
    Score("China", 26, 18, 26), Score("Brasil", 7, 6, 6),
    Score("Hungria", 8, 3, 4), Score("Canadá", 4, 3, 15),
    Score("Japão", 12, 8, 21)]
```

In [1]:
```
from collections import namedtuple
scores = (lambda Score: [
    Score("China", 26, 18, 26), Score("Brasil", 7, 6, 6),
    Score("Hungria", 8, 3, 4), Score("Canadá", 4, 3, 15),
    Score("Japão", 12, 8, 21)]
)(namedtuple("Score", "country gold silver bronze"))
```

Com a ordenação

In [1]:
```
from collections import namedtuple
(lambda scores:
    sorted(scores, reverse=True, key=lambda score: (score.gold,
score.silver, score.bronze, sum(score[1:]), score.country))
)(
    (lambda Score: [
        Score("China", 26, 18, 26), Score("Brasil", 7, 6, 6),
        Score("Hungria", 8, 3, 4), Score("Canadá", 4, 3, 15),
        Score("Japão", 12, 8, 21)]
    )(namedtuple("Score", "country gold silver bronze"))
)
```

Out[1]:
```
[Score(country='China', gold=26, silver=18, bronze=26),
 Score(country='Japão', gold=12, silver=8, bronze=21),
 Score(country='Hungria', gold=8, silver=3, bronze=4),
 Score(country='Brasil', gold=7, silver=6, bronze=6),
 Score(country='Canadá', gold=4, silver=3, bronze=15)]
```

E o import?

In [1]:
```
(lambda scores:
    sorted(scores, reverse=True, key=lambda score: (score.gold,
score.silver, score.bronze, sum(score[1:]), score.country))
)(
    (lambda Score: [
        Score("China", 26, 18, 26), Score("Brasil", 7, 6, 6),
        Score("Hungria", 8, 3, 4), Score("Canadá", 4, 3, 15),
        Score("Japão", 12, 8, 21)]
    )(__import__("collections").namedtuple("Score", "country gold
silver bronze"))
)
```

Out[1]:
```
[Score(country='China', gold=26, silver=18, bronze=26),
 Score(country='Japão', gold=12, silver=8, bronze=21),
 Score(country='Hungria', gold=8, silver=3, bronze=4),
 Score(country='Brasil', gold=7, silver=6, bronze=6),
 Score(country='Canadá', gold=4, silver=3, bronze=15)]
```

Uma linha

```
In [1]: (lambda scores: sorted(scores, reverse=True, key=lambda score:
        (score.gold, score.silver, score.bronze, sum(score[1:]),
        score.country)))((lambda Score: [Score("China", 26, 18, 26),
        Score("Brasil", 7, 6, 6),Score("Hungria", 8, 3, 4), Score("Canadá",
        4, 3, 15),Score("Japão", 12, 8, 21)])(__import__("collections")
        .namedtuple("Score", "country gold silver bronze")))

Out[1]: [Score(country='China', gold=26, silver=18, bronze=26),
         Score(country='Japão', gold=12, silver=8, bronze=21),
         Score(country='Hungria', gold=8, silver=3, bronze=4),
         Score(country='Brasil', gold=7, silver=6, bronze=6),
         Score(country='Canadá', gold=4, silver=3, bronze=15)]
```

Decorator

- Função que recebe função como parâmetro e retorna função alterada

In [1]:
```
def print_values(func):
    def new_func(*args, **kwargs):
        print(args, kwargs)
        result = func(*args, **kwargs)
        print(">", result)
        return result
    return new_func
@print_values
def add(x, y):
    return x + y
add(1, y=2);
```

```
(1,) {'y': 2}
> 3
```

Nome da função (1/2)

```
In [2]: add.__name__

Out[2]: 'new_func'
```

- Nome da função perdido

```
@print_values
def add(x, y):
    return x + y
```

Nome da função (2/2)

In [1]:
```
from functools import wraps
def print_values(func):
    @wraps(func)
    def new_func(*args, **kwargs):
        print(args, kwargs)
        result = func(*args, **kwargs)
        print(">", result)
        return result
    return new_func
...
```

```
...
@print_values
def add(x, y):
    return x + y
add(1, y=2);
```

```
(1,) {'y': 2}
> 3
```

In [2]:
```
add.__name__
```

Out[2]: 'add'

Como funciona um decorator?

In [1]:
```
def add(x, y):
    return x + y
add = print_values(add)
add(1, y=2);
```

```
(1,) {'y': 2}
> 3
```

In [1]:
```
@print_values
def add(x, y):
    return x + y
add(1, y=2);
```

```
(1,) {'y': 2}
> 3
```

Uma linha (1/4)

In [1]:

```
def print_values(func):
    def new_func(*args, **kwargs):
        print(args, kwargs)
        result = func(*args, **kwargs)
        print(">", result)
        return result
    return new_func
@print_values
def add(x, y):
    return x + y
add(1, y=2);
```

```
(1,) {'y': 2}
> 3
```

Uma linha (2/4)

In [1]:

```
def print_values(func):
    new_func = __import__('functools').wraps(func)(
        lambda *args, **kwargs: (
            lambda result: [print(">", result), result][-1]
        )([print(args, kwargs), func(*args, **kwargs)][-1])
    )
    return new_func

def add(x, y):
    return x + y
add = print_values(add)
add(1, y=2);
```

```
(1,) {'y': 2}
> 3
```

Uma linha (3/4)

In [1]:

```
print_values = lambda func: __import__('functools').wraps(func)(
    lambda *args, **kwargs: (
        lambda result: [print(">", result), result][-1]
    )([print(args, kwargs), func(*args, **kwargs)][-1])
)

add = print_values(lambda x, y: x + y)
add(1, y=2);
```

```
(1,) {'y': 2}
> 3
```

Uma linha (4/4)

Instituto de Computação
ESI
Laboratório de Engenharia de Sistemas e Informação

In [1]:
```
(lambda func: __import__('functools').wraps(func)(lambda *args,
**kwargs: (lambda result: [print(">", result), result][-
1])([print(args, kwargs), func(*args, **kwargs)][-1])))(lambda x,
y: x + y)(1, y=2);
```

```
(1,) {'y': 2}
> 3
```

Recursão

- Vamos calcular fibonacci

In [1]:
```
from functools import lru_cache

@lru_cache(maxsize=None)
def fib(n):
    if n <= 1:
        return n
    else:
        return fib(n - 1) + fib(n - 2)

fib(10)
```

Out[1]: 55

Recursão

- Com lambda:

```
In [1]: fib = lambda n: n if n <= 1 else fib(n - 1) + fib(n - 2)
        fib(10)
```

```
Out[1]: 55
```

- Mas isso são duas linhas! Substituindo:

```
In [1]: (lambda n: n if n <= 1 else fib(n - 1) + fib(n - 2))(10)
```

NameError

Solução 1
Passar fib como parâmetro (1/2)

- Com lambda:

In [1]:
```
(lambda fn: fn(fn, 10))(
  lambda fib, n: n if n <= 1 else fib(fib, n - 1) + fib(fib, n - 2)
)
```

Out[1]: 55

- Mantendo a passagem de parâmetro

In [1]:
```
(lambda m: (
  (lambda fn: fn(fn, m))(
  lambda fib, n: n if n <= 1 else fib(fib, n - 1) + fib(fib, n - 2)
  )
))(10)
```

Out[1]: 55

Solução 1
Passar fib como parâmetro (2/2)

- Com lru_cache:

In [1]:
```
(lambda m: (
  (lambda fn: fn(fn, m))(
  __import__('functools').lru_cache(maxsize=None)(
  lambda fib, n: n if n <= 1 else fib(fib, n - 1) + fib(fib, n - 2)
  ))
))(10)
```

Out[1]: 55

Solução 2
Y-Combinator

- https://mvanier.livejournal.com/2897.html
- Y-Combinator é uma função que receve uma função não recursiva e retorna uma versão recursiva dela

In [1]:
```
Y=lambda f:(lambda x:x(x))(lambda y: f(lambda *args: y(y)(*args)))
```

In [2]:
```
func = (
  lambda fib: (lambda n: n if n <= 1 else fib(n - 1) + fib(n - 2))
)
Y(func)(10)
```

Out[2]: 55

Exercícios 4 - 5

4. Faça uma função anônima que faça o mesmo que a seguinte função:

```python
def f(a, b):
    c = a ** 2 + b ** 2
    return c ** 0.5
```

5. Faça uma função que calcule fatorial usando funções anônimas

Mais do functools

- Chamando funções diferentes para tipos diferentes
- Definindo parâmetros padrões

functools.singledispatch

- Chamando funções diferentes para tipos diferentes

In [1]:
```
elements = [2, "a", 4, 1, "b", "c", None]
```

- Quero ordernar deixando 1 próximo do “a”, 2 próximo do “b”, e None no início

singledispatch

In [2]:

```
from functools import singledispatch
@singledispatch
def key(value):
    return value

@key.register(str)
def _(value):
    return ord(value[0].lower()) - ord('a') + 1

@key.register(type(None))
def _(value):
    return -1

key("a"), key(2), key(None)
```

Out[2]: (1, 2, -1)

singledispatch

Instituto de Computação
ESI
Laboratório de Engenharia de Sistemas e Informação

```
In [3]: sorted(elements, key=key)
```

```
Out[3]: [None, 'a', 1, 2, 'b', 'c', 4]
```

Uma linha (1/2)

In [1]:
```
sorted([2, "a", 4, 1, "b", "c", None], key=(lambda key: [
 key.register(str)(lambda value: ord(value[0].lower())-ord('a')+1),
 key.register(type(None))(lambda value: -1),
 key
][-1])(
    __import__("functools").singledispatch(lambda value: value))
)
```

Out[1]: [None, 'a', 1, 2, 'b', 'c', 4]

Uma linha (2/2)

In [1]:
```
sorted([2, "a", 4, 1, "b", "c", None], key=(lambda key:
[key.register(str)(lambda value: ord(value[0].lower())-
ord('a')+1),key.register(type(None))(lambda value: -1),key][-
1])(__import__("functools").singledispatch(lambda value: value)))
```

Out[1]: [None, 'a', 1, 2, 'b', 'c', 4]

functools.partial

- Definindo parâmetros padrões

```
In [1]: int("1001", base=2)

Out[1]: 9
```

- Com partial:

```
In [1]: from functools import partial
        base2 = partial(int, base=2)
        base2("1001")

Out[1]: 9
```

Índice

Estruturas de Dados

- Lista

In [1]:
```
a = [1, 2, 3, 4]
a.append(5)  # O(1) : Inserção no final
a.pop()      # O(1) : Remoção do final
```

Out[1]: 5

In [2]:
```
3 in a       # O(N) : Busca
```

Out[2]: True

In [3]:
```
a[1]         # O(1) : Acesso direto
```

Out[3]: 2

Estruturas de Dados

- Tupla

In [1]:
```
b = 1, 2, 3, 4
# Inserção e Remoção não é possível: tupla é imutável
b
```

Out[1]: (1, 2, 3, 4)

In [2]:
```
3 in b        # O(N) : Busca
```

Out[2]: True

In [3]:
```
b[1]          # O(1) : Acesso direto
```

Out[3]: 2

Estruturas de Dados

- Conjunto

In [1]:
```
c = {1, 2, 3, 4}
c.add(5)      # O(1) : Inserção no conjunto
c.remove(5)   # O(1) : Remoção no conjunto
```

In [2]:
```
3 in c        # O(1) : Busca
```

Out[2]: True

  - Não há acesso por posição

Estruturas de Dados

- Dicionário

In [1]:
```
d = {1: "a", 2: "b", 3: "c", 4: "d"}
d[5] = "e"    # O(1) : Inserção de chave
del d[5]      # O(1) : Remoção de item
```

In [2]:
```
3 in a        # O(N) : Busca de chave
```

Out[2]: True

In [3]:
```
a[1]          # O(1) : Acesso de chave
```

Out[3]: 'a'

Iterando em Estruturas

In [1]:
```
a = [1, 2, 3, 4]
```

In [2]:
```
for element in a:
    print(element)
```

```
1
2
3
4
```

Iterando com Índices

In [1]:
```
a = [1, 2, 3, 4]
```

In [2]:
```
for index, element in enumerate(a):
    print(index, element)
```

```
0 1
1 2
2 3
3 4
```

Iterando em duas estruturas

In [1]:
```
a = [1, 2, 3, 4]
b = [5, 6, 7]
```

In [2]:
```
for a_element, b_element in zip(a, b):
    print(a_element, b_element)
```

```
1 5
2 6
3 7
```

- É possível passar mais do que dois argumentos para o *zip* e iterar em mais do que duas estruturas

itertools.zip_longest

In [1]:
```
a = [1, 2, 3, 4]
b = [5, 6, 7]
```

In [2]:
```
From itertools import zip_longest
for a_element, b_element in zip_longest(a, b):
    print(a_element, b_element)
```

```
1 5
2 6
3 7
4 None
```

- Iterando em duas estruturas, mas indo até o final da maior

Iterando dicionários com chave e valor

In [1]:
```
d = {1: "a", 2: "b", 3: "c", 4: "d"}
```

In [2]:
```
for key, value in d.items():
    print(key, value)
```

```
1 a
2 b
3 c
4 d
```

Transformando Estruturas

map

- Aplica uma função a todos os elementos de um iterável

```
map(func, it)
```

- Retorna um gerador

map

In [1]:
```
a = [1, 2, 3, 4, 5]
```

In [2]:
```
twice_a = map((lambda x: x * 2), a)
twice_a
```

Out[2]:
```
<map at 0x1ab601f7358>
```

In [3]:
```
for element in twice_a:
    print(element)
```

```
2
4
6
8
10
```

map

In [1]:
```
a = [1, 2, 3, 4, 5]
```

In [2]:
```
twice_a = map((lambda x: x * 2), a)
twice_a
```

Out[2]: <map at 0x1ab601f7358>

In [3]:
```
list(twice_a)
```

Out[3]: [2, 4, 6, 8, 10]

filter

- Aplica uma função a todos os elementos de um iterável e retorna os elementos em que a função retorna verdadeiro

```
filter(func, it)
```

- Retorna um gerador

filter

In [1]:
```
a = [1, 2, 3, 4, 5]
```

In [2]:
```
odd = lambda x: x % 2 != 0
odd_a = filter(odd, a)
odd_a
```

Out[2]: <filter at 0x1ab601ac438>

In [3]:
```
list(odd_a)
```

Out[3]: [1, 3, 5]

itertools.filterfalse

- Oposto do *filter*

```
filterfalse(func, it)
```

- Retorna um gerador

itertools.filterfalse

In [1]:
```
a = [1, 2, 3, 4, 5]
```

In [2]:
```
from itertools import filterfalse
odd = lambda x: x % 2 != 0
even_a = filterfalse(odd, a)
even_a
```

Out[2]: <itertools.filterfalse at 0x1ab5fcb9208>

In [3]:
```
list(even_a)
```

Out[3]: [2, 4]

itertools.compress

- Usa outro iterável para selecionar elementos

```
compress(original, selectors)
```

itertools.compress

Instituto de Computação

ESI Laboratório de Engenharia de Sistemas e Informação

In [1]:
```
a = [1, 2, 3, 4, 5]
```

In [2]:
```
from itertools import compress
sub = compress(a, [1, 0, 1, 1, 0])
sub
```

Out[2]: <itertools.compress at 0x1ab5fcb9940>

In [3]:
```
list(sub)
```

Out[3]: [1, 3, 4]

functools.reduce

- Aplica função a dois argumentos cumulativamente da esquerda para a direita

```
reduce(func, it, init)
```

- *init* é opcional. Se não for passado, o valor inicial é o primeiro elemento de *it*
- *func* possui a seguinte assinatura:
  - func(accumulated, current) -> new_accumulated

functools.reduce

In [1]:
```
a = [1, 2, 3, 4, 5]
```

In [2]:
```
from functools import reduce
reduce(lambda acc, element: acc * element, a)
```

Out[2]: 120

In [3]:
```
reduce(lambda acc, element: acc * element, a, 2)
```

Out[3]: 240

Comprehensions

- Açúcar sintático para facilitar transformações de estruturas

List comprehension

- Forma concisa de criar listas baseados em iteráveis

```
[func(x) for x in it if cond(x)]
```

- A parte do *if* é opcional
- É o equivalente a:

```
result = []
for x in it:
    if cond(x):
        result.append(func(x))
```

List comprehension map?

In [1]:
```
a = [1, 2, 3, 4, 5]
```

In [2]:
```
[x * 2 for x in a]
```

Out[2]: [2, 4, 6, 8, 10]

List comprehension filter?

In [1]:
```
a = [1, 2, 3, 4, 5]
```

In [2]:
```
[x for x in a if x % 2 != 0]
```

Out[2]: [1, 3, 5]

List comprehension filterfalse?

In [1]:
```
a = [1, 2, 3, 4, 5]
```

In [2]:
```
[x for x in a if not x % 2 != 0]
```

Out[2]: [2, 4]

List comprehension combinar!

Instituto de Computação

ESI Laboratório de Engenharia de Sistemas e Informação

In [1]:
```
a = [1, 2, 3, 4, 5]
```

In [2]:
```
[x * 2 for x in a if x % 2 != 0]
```

Out[2]: [2, 6]

List comprehension compress?

In [1]:
```
a = [1, 2, 3, 4, 5]
```

In [2]:
```
selectors = [1, 0, 1, 1, 0]
[x for i, x in enumerate(a) if selectors[i]]
```

Out[2]: [1, 3, 4]

In [2]:
```
selectors = [1, 0, 1, 1, 0]
[x for x, use in zip(a, selectors) if use]
```

Out[2]: [1, 3, 4]

List comprehension reduce?

- Não há equivalência

Set comprehension

- Forma concisa de criar conjuntos baseados em iteráveis

```
{func(x) for x in it if cond(x)}
```

- A parte do *if* é opcional
- É o equivalente a:

```
result = set()
for x in it:
    if cond(x):
        result.add(func(x))
```

Set comprehension

```
In [1]: a = [-2, 2, 3, 4]
```

```
In [2]: {x ** 2 for x in a}
```

```
Out[2]: {4, 9, 16}
```

- Note que o resultado possui menos elementos
  - Não há repetição em conjuntos

Dict comprehension

- Forma concisa de criar dicionários baseados em iteráveis

```
{key(x): func(x) for x in it if cond(x)}
```

- A parte do *if* é opcional
- É o equivalente a:

```
result = {}
for x in it:
    if cond(x):
        result[key(x)] = func(x)
```

Dict comprehension

In [1]:
```
d = {1: "a", 2: "b", 3: "c", 4: "d"}
```

In [2]:
```
{value: key for key, value in d.items()}
```

Out[2]: {'a': 1, 'b': 2, 'c': 3, 'd': 4}

Exercícios 6 - 8

6. Faça um dicionário a partir de duas listas de forma que o elemento de uma seja a chave e o elemento da outra seja o valor. Exemplo:

```
l1 = [1, 2, 4, 8]; l2 = ["a", "b", "c", "d"]
esperado == {1: "a", 2: "b", 4: "c", 8: "d"}
```

7. Pegue as letras impares do albeto. Dica:

```
from string import ascii_lowercase
```

8. Transforme uma lista de listas em uma lista só com todos elementos. Exemplo:

```
antes = [[1, 8, 3], [2, 4, 6], [7, 3, 1]]
esperado == [1, 8, 3, 2, 4, 6, 7, 3, 1]
```

~~Tuple comprehension?~~

Instituto de Computação

ESI Laboratório de Engenharia de Sistemas e Informação

In [1]:
```
a = [1, 2, 3, 4, 5]
```

In [2]:
```
(x * 2 for x in a)
```

Out[2]:
```
<generator object <genexpr> at 0x000001AB60056468>
```

In [3]:
```
tuple(x * 2 for x in a)
```

Out[3]:
```
(2, 4, 6, 8, 10)
```

Índice

Geradores

- Funções que se comportam como iteráveis

In [1]:
```
def generate3():
    yield 1
    yield 2
    yield 3
generate3()
```

Out[1]:
```
<generator object generate3 at 0x000001AB60056AF0>
```

In [2]:
```
for element in generate3():
    print(element)
```

```
1
2
3
```

Geradores

```
In [2]: it = iter(generate3())
        next(it)

Out[2]: 1

In [3]: next(it)

Out[3]: 2

In [3]: list(it)

Out[3]: [3]
```

```
def generate3():
    yield 1
    yield 2
    yield 3
```

Generator expression

- Forma concisa de criar geradores de forma semelhante a comprehensions

```
{func(x) for x in it if cond(x)}
```

- A parte do *if* é opcional
- É o equivalente a:

```
def gen(it):
    for x in it:
        if cond(x):
            yield func(x)
gen(it)
```

Generator expression

In [1]:
```
a = [1, 2, 3, 4, 5]
```

In [2]:
```
sum(x ** 2 for x in a)
```

Out[2]: 55

In [3]:
```
gen = (x ** 2 for x in a)
list(gen)
```

Out[3]: [1, 4, 9, 16, 25]

In [4]:
```
list(gen)
```

Out[4]: []

Operações em geradores

- Ou em qualquer iterável
  - Clonar
  - Juntar
  - Slice

itertools.tee

- Clona iterador

```
tee(it, n)
```

- Retorna *n* clones de *it*

itertools.tee

In [1]:
```
a = [1, 2, 3, 4, 5]
```

In [2]:
```
from iterools import tee

gen = (x ** 2 for x in a)
it1, it2 = tee(gen, 2)

list(it1)
```

Out[2]: [1, 4, 9, 16, 25]

In [3]:
```
list(it2)
```

Out[3]: [1, 4, 9, 16, 25]

itertools.chain

- Concaterna iteráveis

```
chain(it1, it2, ..., itn)
```

- Retorna iterador que gera todos elementos it1, it2, ..., itn

itertools.chain

In [1]:
```
a = [1, 2, 3, 4, 5]
```

In [2]:
```
from iterools import chain

gen1 = (x ** 2 for x in a)
gen2 = (x ** 3 for x in a)

list(chain(gen1, gen2))
```

Out[2]: [1, 4, 9, 16, 25, 1, 8, 27, 64, 125]

itertools.islice

- Slice de iterável

```
islice(it, start=0, end=None, step=1]
```

- Faz o equivalente a it[start:end:step]
  - Funciona em qualquer iterador
  - Step não pode ser negativo

itertools.islice

In [1]:
```
a = [1, 2, 3, 4, 5]
```

In [2]:
```
from iterools import chain

gen1 = (x ** 2 for x in a)
gen2 = (x ** 3 for x in a)
gen = chain(gen1, gen2)

list(islice(gen, 2, None, 2))
```

Out[2]: [9, 25, 8, 64]

Geradores infinitos

- Geradores que não terminam
- Não use em funções que tentam consumir todo o iterável!
  - list(infinito)
- Do itertools
  - count
  - cycle
  - repeat

itertools.count

- Conta a partir de um número

```
count(start, step=1)
```

- Retorna

```
start
start + step
start + 2 * step
start + 3 * step
...
```

itertools.count

```
In [1]: from iterools import count

        gen = count(50)

        list(zip(gen, range(10)))
```

```
Out[1]: [(50, 0),
         (51, 1),
         (52, 2),
         (53, 3),
         (54, 4),
         (55, 5),
         (56, 6),
         (57, 7),
         (58, 8),
         (59, 9)]
```

zip e *range(10)* sendo usados para limitar tamanho do resultado

itertools.cycle

- Repete iterável

```
cycle(p)
```

- Retorna

```
p[0]
p[1]
...
p[-1]
p[0]
p[1]
...
```

itertools.cycle

In [1]:
```
from iterools import cycle

gen = cycle('abc')

list(zip(gen, range(10)))
```

Out[1]:
```
[('a', 0),
 ('b', 1),
 ('c', 2),
 ('a', 3),
 ('b', 4),
 ('c', 5),
 ('a', 6),
 ('b', 7),
 ('c', 8),
 ('a', 9)]
```

zip e *range(10)* sendo usados para limitar tamanho do resultado

itertools.repeat

- Repete valor

```
repeat(e)
```

- Retorna

```
e
e
e
e
...
```

- Pode ser limitado por atributo adicional

```
repeat(e, n)
```

itertools.repeat

In [1]:
```
from iterools import repeat
gen = repeat('a')
list(zip(gen, range(10)))
```

Out[1]:
```
[('a', 0),
 ('a', 1),
 ('a', 2),
 ('a', 3),
 ('a', 4),
 ('a', 5),
 ('a', 6),
 ('a', 7),
 ('a', 8),
 ('a', 9)]
```

zip e *range(10)* sendo usados para limitar tamanho do resultado

In [2]:
```
list(repeat('a', 10))
```

Out[2]:
```
['a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a']
```

Exemplo mais complicado

- Sequência de collatz

In [1]:
```
def next_collatz(x):
    return x // 2 if x % 2 == 0 else 3 * x + 1

def collatz(current):
    while current != 1:
        yield current
        current = next_collatz(current)
    yield current

list(collatz(5))
```

Out[1]: [5, 16, 8, 4, 2, 1]

Uma linha (1/2)

- Sequência de collatz

In [1]:
```
from itertools import accumulate, takewhile, chain, repeat

next_collatz = lambda x: x // 2 if x % 2 == 0 else 3 * x + 1
collatz_acc = lambda acc, new: next_collatz(acc)
collatz = lambda c: chain(
  takewhile(
    (lambda x: x != 1),
    accumulate(repeat(c), collatz_acc)
  ),
  [1]
)
list(collatz(5))
```

Out[1]: [5, 16, 8, 4, 2, 1]

Uma linha (2/2)

- Sequência de collatz

In [1]:
```
list((lambda c:__import__('itertools').chain(__import__('itertools'
).takewhile((lambda x:x!=1),__import__('itertools').accumulate
(__import__('itertools').repeat(c), (lambda acc, new: (lambda x: x
// 2 if x % 2 == 0 else 3 * x + 1)(acc)))),[1]))(5))
```

Out[1]: [5, 16, 8, 4, 2, 1]

itertools.accumulate

- Gera valores acumulados

```
accumulate(it, func)
```

- Equivale a

```
def accumulate(it, func):
    it_iter = iter(it)
    total = next(it_iter)
    yield total
    for x in it_iter:
        total = func(total, x)
        yield total
```

itertools.accumulate (No exemplo)

Instituto de Computação
ESI Laboratório de Engenharia de Sistemas e Informação

In [2]:
```
list(accumulate([5, 0, 0, 0, 0, 0], collatz_acc))
```

Out[2]: [5, 16, 8, 4, 2, 1]

In [3]:
```
list(accumulate([5, 0, 0, 0, 0, 0, 0, 0], collatz_acc))
```

Out[3]: [5, 16, 8, 4, 2, 1, 4, 2]

In [4]:
```
list(takewhile(
  (lambda x: x != 1),
  accumulate(repeat(5), collatz_acc)
))
```

Out[4]: [5, 16, 8, 4, 2]

```
next_collatz = lambda x: x // 2 if x % 2 == 0 else 3 * x + 1
collatz_acc = lambda acc, new: next_collatz(acc)
```

itertools.takewhile

- Gera valores acumulados

```
takewhile(func, it)
```

- Equivale a

```
def takewhile(func, it):
    for x in it:
        if not func(x):
            break
        yield x
```

itertools.takewhile

In [1]:
```
from itertools import takewhile
list(takewhile(lambda x: x != 1, [5, 16, 8, 4, 2, 1, 4, 2]))
```

Out[1]: [5, 16, 8, 4, 2]

itertools.dropwhile

- Pega todos elementos depois da condição ter sido satisfeita

```
dropwhile(func, it)
```

- Equivale a

```
def dropwhile(func, it):
    for x in it:
        if func(x):
            yield x
            break
    yield from it
```

itertools.dropwhile

In [1]:
```
from itertools import dropwhile
list(dropwhile(lambda x: x != 1, [5, 16, 8, 4, 2, 1, 4, 2]))
```

Out[1]: [1, 4, 2]

Exercícios 9 - 11

9. Crie uma função que acesse a n-ésima posição de um gerador

```
access((x + 2 for x in [1, 2, 4, 8]), 3) == 10
```

10. Faça um gerador para a sequencia de Fibonacci
11. Substitua o gerador por um feito com accumulate

Índice

Mais do itertools

- Agrupando valores
- Aplicando funções a sub-iteráveis
- Combinatória

itertools.groupby

- Agrupa valores ordenados formando um iterável de iteráveis

```
groupby(it, func)
```

Exige que o iterável original esteja ordenado pelo critério de agrupamento

itertools.groupby

In [1]:
```
a = [1, 2, 3, 4, 5, 6, 7, 8]
key = lambda x: x % 2
b = sorted(a, key=key)
b
```

Out[1]: [2, 4, 6, 8, 1, 3, 5, 7]

In [2]:
```
from itertools import groupby
group = groupby(b, key)
group
```

Out[2]: <itertools.groupby at 0x1ab600a2188>

In [3]:
```
for remainder, numbers in group:
    print(remainder, list(numbers))
```

```
0 [2, 4, 6, 8]
1 [1, 3, 5, 7]
```

itertools.starmap

- Aplica função a sub-iteráveis

```
starmap(func, it)
```

itertools.starmap

In [1]:
```
a = [(2, 5), (3, 2), (10, 3)]

def half_of_pow(x, y):
    return (x ** y) / 2
```

In [2]:
```
from itertools import starmap
list(starmap(half_of_pow, a))
```

Out[2]: [16.0, 4.5, 500.0]

Combinatória

- Produto
- Permutações
- Combinações
- Combinações com repetição

itertools.product

- Equivale a for alinhado

```
product(p, q, ..., repeat=1)
```

- Parâmetro repeat indica número de vezes que *p, q* e outros iteráveis são repetidos

itertools.product

In [1]:
```
from itertools import product
list(product("ABCD", "EFGH"))
```

Out[1]:
```
[('A', 'E'), ('A', 'F'),
 ('A', 'G'), ('A', 'H'),
 ('B', 'E'), ('B', 'F'),
 ('B', 'G'), ('B', 'H'),
 ('C', 'E'), ('C', 'F'),
 ('C', 'G'), ('C', 'H'),
 ('D', 'E'), ('D', 'F'),
 ('D', 'G'), ('D', 'H')]
```

itertools.product

In [1]:
```
from itertools import product
list(product([False, True], repeat=3))
```

Out[1]:
```
[(False, False, False),
 (False, False, True),
 (False, True, False),
 (False, True, True),
 (True, False, False),
 (True, False, True),
 (True, True, False),
 (True, True, True)]
```

itertools.permutations

- Gera todas as permutações de tamanho *r* sem repetição

```
permutations(p, r=len(p))
```

itertools.permutations

In [1]:
```
from itertools import permutations
list(permutations("ABC"))
```

Out[1]:
```
[('A', 'B', 'C'),
 ('A', 'C', 'B'),
 ('B', 'A', 'C'),
 ('B', 'C', 'A'),
 ('C', 'A', 'B'),
 ('C', 'B', 'A')]
```

itertools.permutations

In [1]:
```
from itertools import permutations
list(permutations("ABC", 2))
```

Out[1]:
```
[('A', 'B'),
 ('A', 'C'),
 ('B', 'A'),
 ('B', 'C'),
 ('C', 'A'),
 ('C', 'B')]
```

itertools.combinations

- Gera todas as combinações (ordenadas) de tamanho *r* sem repetição

```
combinations(p, r)
```

itertools.combinations

Instituto de Computação
ESI
Laboratório de Engenharia de Sistemas e Informação

In [1]:
```
from itertools import combinations
list(combinations("ABC", 2))
```

Out[1]:
```
[('A', 'B'),
 ('A', 'C'),
 ('B', 'C')]
```

itertools. combinations_with_replacement

- Gera todas as combinações (ordenadas) de tamanho *r* com repetição

```
Combinations_with_replacement(p, r)
```

itertools.
combinations_with_replacement

In [1]:
```
from itertools import combinations_with_replacement
list(combinations_with_replacement("ABC", 2))
```

Out[1]:
```
[('A', 'A'),
 ('A', 'B'),
 ('A', 'C'),
 ('B', 'B'),
 ('B', 'C'),
 ('C', 'C')]
```

Exercício 12

12. Obtenha todas as permutações "ABCD" que tenham "D" como último elemento

Mais conteúdo

- Process of Python mastery: https://stackoverflow.com/questions/2573135/python-progression-path-from-apprentice-to-guru/2576240#2576240
- Onelinerizer: http://www.onelinerizer.com/
  - Ferramenta automática que faz o que foi apresentado
  - O site apresenta uma forma mais “algorítmica” de fazer a conversão

Índice functools

- cmp_to_key
- wraps
- lru_cache
- singledispatch
- partial
- reduce

Índice itertools

- zip_longest
- filterfalse
- compress
- tee
- chain
- islice
- count
- cycle
- repeat
- accumulate
- takewhile
- dropwhile
- groupby
- starmap
- product
- permutations
- combinations
- combinations_with_replacement

Python em Uma Linha

https://github.com/JoaoFelipe/presentations/

João Felipe N. Pimentel

joaofelipenp@gmail.com

Instituto de Computação

Laboratório de Engenharia de Sistemas e Informação