Geek University

Evolua seu lado geek!

www.geekuniversity.com.br

Conhecendo a API de Multiprocessamento do Python

Conhecendo a API de Multiprocessamento do Python

Uma das coisas que fazem a computação, em geral, ter sucesso é a padronização.

Conhecendo a API de Multiprocessamento do Python

Por exemplo, se analisarmos uma implementação simples de Thread em Python temos:

```python
import threading


def faz_algo(valor):
    print(f'Fazendo algo com o {valor}')




def main():
    th = threading.Thread(target=faz_algo, args=('Pássaro',))

    th.start()
    th.join()




if __name__ == '__main__':
    main()

```

Conhecendo a API de Multiprocessamento do Python

Compare com a implementação simples de Processos com Python e veja o padrão se repetir:

```python
import threading


def faz_algo(valor):
    print(f'Fazendo algo com o {valor}')




def main():
    th = threading.Thread(target=faz_algo, args=('Pássaro',))

    th.start()
    th.join()



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

```python
import multiprocessing


def faz_algo(valor):
    print(f'Fazendo algo com o {valor}')




def main():
    pc = multiprocessing.Process(target=faz_algo, args=('Pássaro',))

    pc.start()
    pc.join()



if __name__ == '__main__':
    main()

```

Conhecendo a API de Multiprocessamento do Python

Compare com a implementação simples de Processos com Python e veja o padrão se repetir:

```python
import threading


def faz_algo(valor):
    print(f'Fazendo algo com o {valor}')




def main():
    th = threading.Thread(target=faz_algo, args=('Pássaro',))

    th.start()
    th.join()



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

```python
import multiprocessing


def faz_algo(valor):
    print(f'Fazendo algo com o {valor}')




def main():
    pc = multiprocessing.Process(target=faz_algo, args=('Pássaro',))

    pc.start()
    pc.join()



if __name__ == '__main__':
    main()

```

Esta padronização nos ajuda a alternar entre uma API e outra e realizarmos testes de performance ou qualquer outra coisa de maneira fácil.

Conhecendo a API de Multiprocessamento do Python

Desta forma você pode imaginar que o construtor da classe Thread...

```
class Thread:
    """A class that represents a thread of control.

    This class can be safely subclassed in a limited fashion. There are two ways
    to specify the activity: by passing a callable object to the constructor, or
    by overriding the run() method in a subclass.

    """

    _initialized = False

    def __init__(self, group=None, target=None, name=None,
                 args=(), kwargs=None, *, daemon=None):
        """This constructor should always be called with keyword arguments. Arguments are:
```

Conhecendo a API de Multiprocessamento do Python

...é parecido com o construtor da classe Process, tirando o fato da classe Process herdar da classe BaseProcess...

```
class BaseProcess(object):
    '''
    Process objects represent activity that is run in a separate process

    The class is analogous to `threading.Thread`
    '''
    def _Popen(self):
        raise NotImplementedError

    def __init__(self, group=None, target=None, name=None, args=(), kwargs={},
                 *, daemon=None):
```

Conhecendo a API de Multiprocessamento do Python

...é parecido com o construtor da classe Process, tirando o fato da classe Process herdar da classe BaseProcess...

```
class BaseProcess(object):
    '''
    Process objects represent activity that is run in a separate process

    The class is analogous to `threading.Thread`
    '''
    def _Popen(self):
        raise NotImplementedError

    def __init__(self, group=None, target=None, name=None, args=(), kwargs={},
                 *, daemon=None):
```

Vamos ao código criar nosso primeiro processo…

Geek University

Evolua seu lado geek!

www.geekuniversity.com.br