Executando programas externos com o módulo *subprocess*

Msc. João Gabriel Rodinho Nunes Ferreira

V Workshop de Python para Dados Biológicos
26-30 de setembro 2022

Motivação

Quero utilizar um programa dentro do meu script em Python

ex: montador de genomas

Porém, esse programa não foi escrito em Python.

Não posso diretamente importar suas funções e utilizá-las :(

Mas posso utilizar o módulo subprocess para executar o programa :)

Conteúdo

Introdução teórica

Mão na massa: explorando a função *run()*

Salvando os outputs dos subprocessos

Exemplo de aplicação do módulo subprocess: MitoHiFi

Conteúdo

Introdução teórica

Mão na massa: explorando a função *run()*

Salvando os outputs dos subprocessos

Exemplo de aplicação do módulo subprocess: MitoHiFi

Subprocess

O módulo *subprocess* permite a criação de novos processos, a conexão aos seus *input/output/mensagens de erro*, e a obtenção dos seus códigos de retorno.

https://docs.python.org/3/library/subprocess.html

Subprocess

O módulo *subprocess* permite a criação de novos processos, a conexão aos seus *input/output/mensagens de erro*, e a obtenção dos seus códigos de retorno.

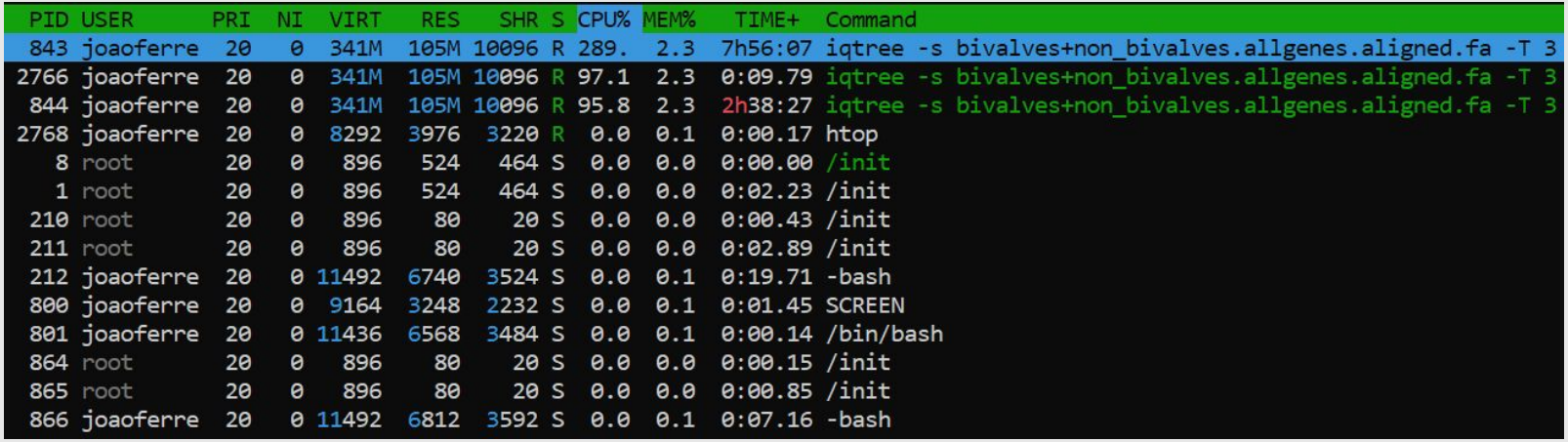
https://docs.python.org/3/library/subprocess.html

Processos

Um processo é uma abstração do sistema operacional de um programa em execução.

Sistema operacional

Tipicamente, centenas a milhares de processos são executados de forma simultânea pelo sistema operacional.



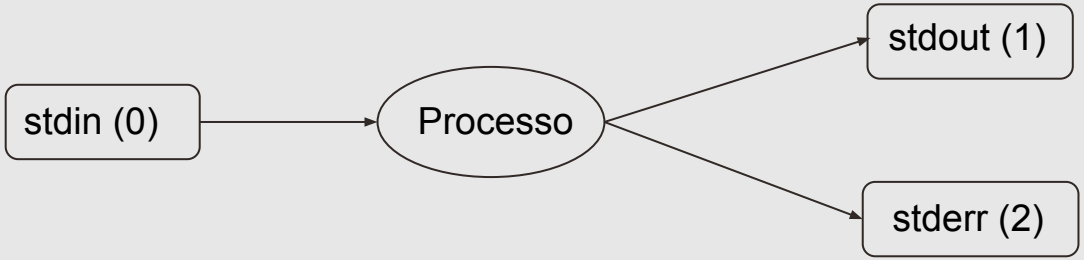
PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
843	joaoferre	20	0	341M	105M	10096	R	289.	2.3	7h56:07	iqtree -s bivalves+non_bivalves.allgenes.aligned.fa -T 3
2766	joaoferre	20	0	341M	105M	10096	R	97.1	2.3	0:09.79	iqtree -s bivalves+non_bivalves.allgenes.aligned.fa -T 3
844	joaoferre	20	0	341M	105M	10096	R	95.8	2.3	2h38:27	iqtree -s bivalves+non_bivalves.allgenes.aligned.fa -T 3
2768	joaoferre	20	0	8292	3976	3220	R	0.0	0.1	0:00.17	htop
8	root	20	0	896	524	464	S	0.0	0.0	0:00.00	/init
1	root	20	0	896	524	464	S	0.0	0.0	0:02.23	/init
210	root	20	0	896	80	20	S	0.0	0.0	0:00.43	/init
211	root	20	0	896	80	20	S	0.0	0.0	0:02.89	/init
212	joaoferre	20	0	11492	6740	3524	S	0.0	0.1	0:19.71	-bash
800	joaoferre	20	0	9164	3248	2232	S	0.0	0.1	0:01.45	SCREEN
801	joaoferre	20	0	11436	6568	3484	S	0.0	0.1	0:00.14	/bin/bash
864	root	20	0	896	80	20	S	0.0	0.0	0:00.15	/init
865	root	20	0	896	80	20	S	0.0	0.0	0:00.85	/init
866	joaoferre	20	0	11492	6812	3592	S	0.0	0.1	0:07.16	-bash

Subprocess

O módulo *subprocess* permite a criação de novos processos, a conexão aos seus input/output/mensagens de erro, e a obtenção dos seus códigos de retorno.

https://docs.python.org/3/library/subprocess.html

Fluxos padrão (standard streams)



Subprocess

O módulo *subprocess* permite a criação de novos processos, a conexão aos seus *input/output/mensagens de erro*, e a obtenção dos seus códigos de retorno.

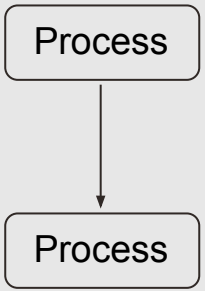
https://docs.python.org/3/library/subprocess.html

Códigos de retorno (*return code | exit status*)

Ao final de sua execução, processos devem informar códigos de retorno sinalizando o seu *status* de saída/conclusão

return code = 0 —-> sucesso

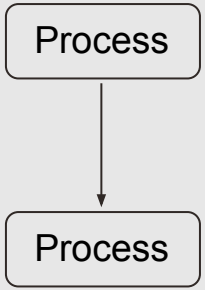
return code ≠ 0 —-> falha



pai

filho (subprocesso)

Subprocesso



pai python

filho subprocesso criado pelo programador

Conteúdo

Introdução teórica

Mão na massa: explorando a função run()

Salvando os outputs dos subprocessos

Exemplo de aplicação do módulo subprocess: MitoHiFi

subprocess.*run()*

- Executa o comando solicitado, espera até o comando finalizar e retorna uma instância *CompletedProcess*
- Função padrão e recomendada para criação de subprocessos
- Deve ser utilizada sempre que possível
- Para casos onde a função *run()* não resolva, a classe *Popen* pode ser utilizada

subprocess.*run*(*args*, *, *stdin=None, input=None, stdout=None, stderr=None, capture_output=False, shell=False, cwd=None, timeout=None, check=False, encoding=None, errors=None, text=None, env=None, universal_newlines=None, **other_popen_kwargs*)

python=3.10.6

subprocess.*run*(*args*, *, stdin=None, input=None, stdout=None, stderr=None, capture_output=False, shell=False, cwd=None, timeout=None, check=False, encoding=None, errors=None, text=None, env=None, universal_newlines=None, **other_popen_kwargs*)**

python=3.10.6

subprocess.*run*(*args*, *, stdin=None, input=None, stdout=None, stderr=None, capture_output=False, shell=False, cwd=None, timeout=None, check=False, encoding=None, errors=None, text=None, env=None, universal_newlines=None, **other_popen_kwargs*)**

python=3.10.6

subprocess.*run*(*args*, *, stdin=None, input=None, stdout=None, stderr=None, capture_output=False, shell=False, cwd=None, timeout=None, check=False, encoding=None, errors=None, text=None, env=None, universal_newlines=None, **other_popen_kwargs*)**

python=3.10.6

Conteúdo

Introdução teórica

Mão na massa: explorando a função run()

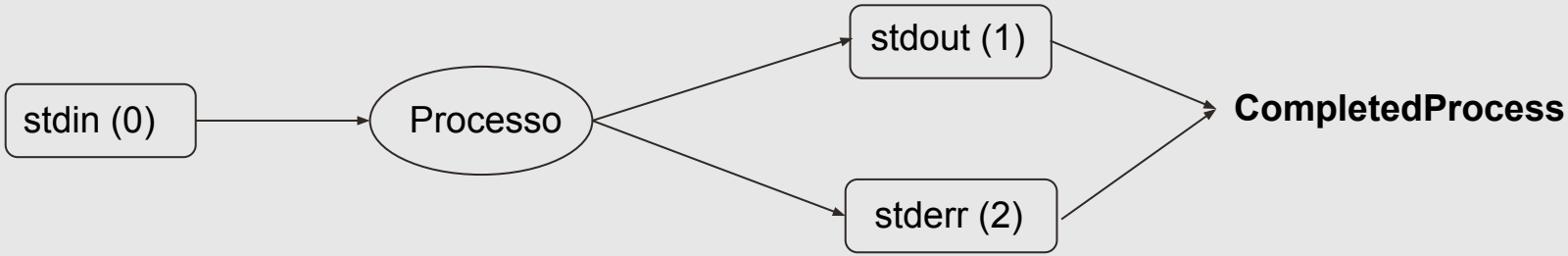
Salvando os outputs dos subprocessos

Exemplo de aplicação do módulo subprocess: MitoHiFi

Salvando os outputs dos subprocessos

Até o momento aprendemos a criar novos subprocessos, mas o output desses processos está sendo impresso no terminal.

Como salvar o output desses processos para que possamos utilizá-lo dentro do nosso script em Python?



subprocess.*run(*
args, *, stdin=None, input=None, stdout=None, stderr=None,
capture_output=False, shell=False, cwd=None, timeout=None, check=False, encoding=None, errors=None, text=None, env=None, universal_newlines=None, **other_popen_kwargs)

Ao setarmos o argumento **capture_output=True**, a função *run()* irá redirecionar os STDOUT e o STDERR do subprocesso executado para o objeto CompletedProcess.

subprocess.*run(*
args, *, stdin=None, input=None, stdout=None, stderr=None, capture_output=False, shell=False, cwd=None, timeout=None, check=False, encoding=None, errors=None, text=None, env=None, universal_newlines=None, **other_popen_kwargs)

Ao setarmos o argumento **text=True**, a função *run()* irá automaticamente decodificar o output de bytes para texto.

python=3.10.6

subprocess.*run(*
args, *, stdin=None, input=None, stdout=None, stderr=None, capture_output=False, shell=False, cwd=None, timeout=None, check=False,
encoding=*None, errors=None, text=None, env=None, universal_newlines=None, **other_popen_kwargs)*

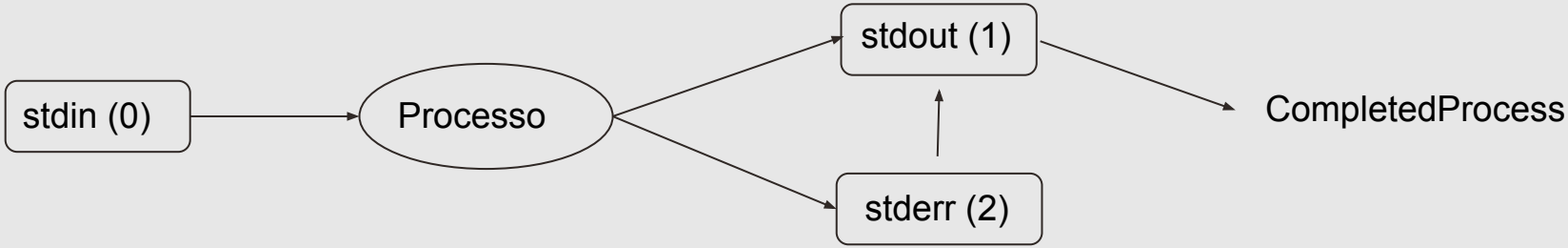
O argumento **encoding** permite definirmos explicitamente uma codificação (ex: “utf-8”) e deve ser privilegiado em relação ao argumento **text**.

python=3.10.6

subprocess.*run*(
***stdout=*None, *stderr=*None, *capture_output=False, shell=False, cwd=None, timeout=None, check=False, encoding=None, errors=None, text=None, env=None, universal_newlines=None, **other_popen_kwargs*)**

Os argumentos **stdout** e **stderr** podem ser utilizados para definir explicitamente onde iremos salvar os conteúdos gerados pelo subprocesso.

É possível salvar em STDOUT e STDERR juntos?



subprocess.*run*(*args*, *, *stdin*=None, *input*=None, *stdout*=None, *stderr*=None, *capture_output*=False, *shell*=False, *cwd*=None, *timeout*=None, *check*=False, *encoding*=None, *errors*=None, *text*=None, *env*=None, *universal_newlines*=None, ***other_popen_kwargs*)

É possível redirecionar os conteúdos de stdout e stderr para arquivos.

python=3.10.6

Conteúdo

Introdução teórica

Mão na massa: explorando a função run()

Salvando os outputs dos subprocessos

Exemplo de aplicação do módulo subprocess: MitoHiFi

MitoHiFi:

Uma pipeline em Python para montagem e anotação de genomas mitocondriais

https://github.com/marcelauliano/MitoHiFi

MitoHiFi

------ This is v2.3 -------

MitoHiFi is a python pipeline distributed under the MIT License

MitoHiFi was first developed to assemble the mitogenomes for a wide range of species in the Darwin Tree of Life Project (DToL)

Dra. Marcela Uliano-Silva

Msc. João Ferreira

MitoHiFi:

subprocess.run() utilizada em 8 etapas

- Mapeamento de leituras de sequenciamento (programa minimap2)
- Montagem das sequências (programa hifiasm)
- Identificação de sequências com características mitocondriais (programa blast)

MitoHiFi:

mitohifi.py

(...)

```
hifiasm_cmd = ["hifiasm", "--primary", "-t", str(args.t), "-f", str(args.m),
                        "-o", "gbk.HiFiMapped.bam.filtered.assembled",
                        "gbk.HiFiMapped.bam.filtered.fasta"]

with open("hifiasm.log", "w") as hifiasm_log_f:
        subprocess.run(hifiasm_cmd, stderr=subprocess.STDOUT, stdout=hifiasm_log_f)
```

(...)

Quando utilizar subprocess (e quando não)

- Execução de programas mais especializados, cujas tarefas não são cobertas por bibliotecas disponíveis em Python. Ex: programas de bioinformática

- Execução de tarefas comuns, que provavelmente podem ser realizadas utilizando alguma biblioteca Python disponível. Ex:

  subprocess.run(“ls”) vs. os.listdir()

Recapitulando...

- Introdução a (sub)processos e fluxos padrões de dados
- Execução de processos com a função subprocess.run()
- Salvando outputs de processos
  - Dentro do script em Python
  - Em arquivos
- Exemplo prático de utilização do módulo subprocess na bioinformática

Explore a documentação oficial

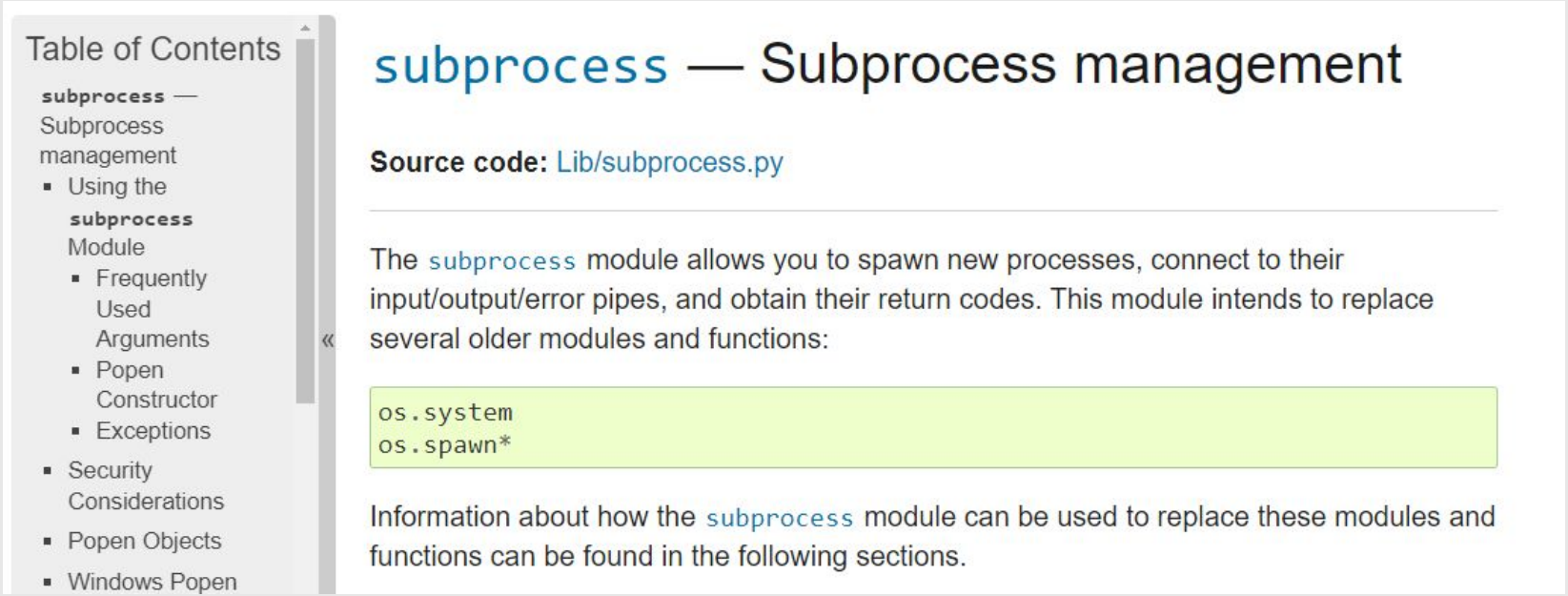


Table of Contents

- **subprocess** — Subprocess management
  - Using the **subprocess** Module
    - Frequently Used Arguments
    - Popen Constructor
    - Exceptions
  - Security Considerations
  - Popen Objects
  - Windows Popen

subprocess — Subprocess management

Source code: Lib/subprocess.py

The `subprocess` module allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes. This module intends to replace several older modules and functions:

```
os.system
os.spawn*
```

Information about how the `subprocess` module can be used to replace these modules and functions can be found in the following sections.

https://docs.python.org/3/library/subprocess.html

Obrigado!

Perguntas?

ferreiranunesjoao@gmail.com
@ferreiranunesj (Twitter)

Redirecionando os fluxos de dados entre subprocessos

A **pipe** é um elemento que permite que os dados de saída (output) de um processo sejam redirecionados como dados de entrada (input) de um outro processo.