PYTHON PARA SYSADMIN

A elección entre Python, Bash PowerShell para administrar sistemas depende de varios factores, incluíndo a complexidade das tarefas de administración.

De seguido algunhas razóns polas cales Python pode ser elixido para administración de sistemas no lugar de Bash ou PowerShell:

- **Flexibilidade e portabilidade**: pódense escribir *scripts* para unha gran cantidade de tarefas como análise de datos, desenvolvemento web, etc. incluíndo tarefas de administración de sistemas. Ademais é **compatible entre plataformas e sistemas operativos**.
- **Gran cantidade de bibliotecas**: Python conta cunha gran variedade de bibliotecas e módulos deseñados para administrar sistemas.
- **Lexibilidade e mantenibilidade**: sintaxe clara e lexible que facilita o mantemento de *scripts* a longo prazo.
- **Comunidade e soporte**: gran comunidade de programadores e grandes recursos dispoñibles, incluíndo documentación.

Todas as **librerías que se estudan neste tema son librerías do sistema**. Existen máis librerías que proporcionan funcionalidades adicionais e facilitar certas tarefas.

1. BIBLIOTECA OS

A biblioteca `os` de Python proporciona funcións para interactuar co sistema operativo no que se está executando o *script* de Python. Algunhas funcionalidades que nos proporciona dita librería.

- Interacción co sistema de ficheiros.
- Acceso a variables de contorno.
- Operacións de ruta
- Acceso a información do sistema

Esta biblioteca de Python é unha ferramenta útil para interactuar co sistema operativo, o que o fai útil para unha gran parte das tarefas de administración de sistema e automatización de tarefas.

1.1. DETECTAR O SISTEMA OPERATIVO

Podemos obter información sobre o sistema operativo no que se execute o *script* utilizando os seguintes elementos:

- `os.name`: este devolve o valor do sistema operativo. Devolverá “posix” para sistemas tipo Unix (Linux, macos) e “nt” para “Windows”.
- `os.uname()`: esta función proporciona información máis detallada sobre o sistema operativo en sistemas Unix. Devolve unha tupla que contén información como o nome do sistema operativo, o nome de equipo, a versión do sistema operativo, a arquitectura hardware e outros detalles específicos do sistema.

Exemplo

```
import os

# Obtenr o nome do sistema operativo
nome_sistema_operativo = os.name
print("Nome do sistema operativo:", nome_sistema_operativo)

# Obtener información detallada sobre el sistema operativo (solo en sistemas Unix)
if nome_sistema_operativo == 'posix':
    info_sistema = os.uname()
    print("Información detallada do sistema operativo:", info_sistema)
```

1.2. VARIABLES DE CONTORNO

Para acceder a variables de contorno podemos utilizar a función `os.getenv(variable)`. Esta función recibe como argumento unha cadea de caracteres co nome da variable de contorno da que se desexa obter o valor. Se a variable non existe, devolverase o valor `None`.

```
import os

# Obter o valor da variable de contorno PATH
valor_path = os.getenv("PATH")

print("Valor da variable de contorno PATH:", valor_path)
```

1.2.1. Engadir variables de contorno

Para crear unha variable de contorno en Python podemos utilizar a función `os.putenv(variable, valor)`. Esta recibe como primeiro parámetro o valor da variable de contorno e como segundo argumento o valor que se lle desexa asignar.

```
import os

# Establecer unha variable de contorno chamada "VARIABLE_ PROBA"
os.putenv("VARIABLE_ PROBA", "valor_de_proba")

# Verificar que la variable de entorno se ha establecido correctamente
print("Variable de contorno VARIABLE_ PROBA establecida:", os.getenv("VARIABLE_ PROBA"))
```

1.3. RUTAS SISTEMA DE FICHEIROS

1.3.1. Compoñer rutas

A función `os.path.join(dir1, dir2, dir3,...)` utilízase para unir un ou máis compoñentes dunha ruta de xeito seguro utilizando o separador de ruta apropiado para o sistema operativo no que se executa o *script*. Isto é útil para construír rutas de arquivos ou directorios e xeito que sexan portables entre diferentes sistemas operativos.

Esta función toma como argumentos os elementos da ruta que se van a unir.

```
import os

# Unir elementos de ruta para formar unha ruta completa
ruta_completa = os.path.join("directorio", "subdirectorio", "arquivo.txt")
print("Ruta completa:", ruta_completa)
```

Este código imprimirá “Ruta completa: directorio/subdirectorio/arquivo.txt” en sistemas Unix e “Ruta completa: directorio\subdirectorio\archivo.txt” en Windows.

1.3.2. Obter nome de ficheiro

A función `os.path.basename(ruta)` utilízase para obter o nome dun ficheiro ou directorio a partir dunha ruta. Por exemplo se temos a ruta “/home/usuario/arquivo.txt” esta función devolverá “arquivo.txt”.

```
import os

ruta = "/home/usuario/arquivo.txt"

nome_arquivo = os.path.basename(ruta)

print("Nome do arquivo:", nome_arquivo)
```

1.3.3. Obter directorio contedor

A función `os.path.dirname(ruta)` utilízase para obter un directorio pai dunha ruta de arquivo.

```
import os

ruta = "/home/usuario/arquivo.txt"

directorio_pai = os.path.dirname(ruta)

print("Directorio pai:", directorio_pai) # Imprime "/home/usuario"
```

1.3.4. Obter ruta absoluta

A función `os.path.abspath(ruta)` utilízase para obter a ruta absoluta correspondente a unha ruta relativa.

```
import os

ruta_relativa = "arquivos/datos.txt"
```

```
ruta_absoluta = os.path.abspath(ruta_relativa)

print("Ruta absoluta:", ruta_absoluta) # imprime por exemplo
/home/usuario/arquivos/datos.txt
```

1.4. COMPROBACIÓNS SISTEMA DE FICHEIROS

1.4.1. Comprobar a existencia

A función `os.path.exists(path)` utilízase para verificar se unha ruta existe nun sistema de arquivos. Devolve verdadeiro se existe e falso en caso contrario.

```
import os

# Verificar se o arquivo "arquivo.txt" existe
if os.path.exists("arquivo.txt"):
    print("O arquivo existe.")
else:
    print("O arquivo non existe.")
```

1.4.2. Comprobar se é un ficheiro

Podemos utilizar a función `os.path.isfile(path)` para verificar se unha ruta dada é un arquivo. Esta función devolve verdadeiro en caso que así sexa e falso en caso contrario.

```
import os

# Ruta do arquivo que se desexa verificar
ruta = "/ruta/al/arquivo.txt"

# Verificar si la ruta es un archivo
if os.path.isfile(ruta):
    print("A ruta corresponde é un archivo.")
else:
    print("A ruta non se corresponde cun archivo.")
```

1.4.3. Comprobar se é un directorio

para verificar se unha ruta é un directorio podemos utilizar a función `os.path.isdir(path)`. esta función devolve `True` en caso afirmativo e `False` en caso negativo.

```
import os

# Ruta do directorio que se desexa verificar
ruta = "/ruta/ao/directorio"

# Verificar se a ruta é un directorio
if os.path.isdir(ruta):
    print("A ruta corresponde a un directorio.")
else:
    print("A ruta non corresponde a un directorio.")
```

1.5. PROPIEDADES SISTEMA DE FICHEIROS

1.5.1. Obter data de modificación do ficheiro

A función `os.path.getmtime(path)` utilízase para obter a data de modificación dun arquivo ou directorio en forma de *timestamp*.

Con `datetime.fromtimestamp()` podemos converter ese valor a un obxecto `datetime`.

```
import os
from datetime import datetime

ruta_arquivo = 'arquivo.txt'

data_modificacion = os.path.getmtime(ruta_arquivo)

data_modificacion_dt = datetime.fromtimestamp(data_modificacion)
```

Para obter a fecha de creación en Windows podémolo facer con `os.path.getctime()`. En sistemas Linux ou macOS a data de creación non está directamente dispoñible, pero podemos utilizar `os.path.getmtime()` como un substituto aproximado para a data de creación.

1.5.2. Obter tamaño

A función `os.path.getsize(path)` utilízase para obter o **tamaño en bytes** dun ficheiro.

```
import os

ruta_arquivo = 'archivo.txt'

tamano_arquivo_bytes = os.path.getsize(ruta_arquivo)
```

Se se lle pasa a **ruta dun directorio**, obterase o tamaño de tódolos ficheiros de ese directorio e de tódolos seus directorios de xeito recursivo.

1.5.3. Obter propiedades

Para obter as propiedades dun ficheiro ou directorio podemos utilizar a función `os.stat(path)`. Este devolve un obxecto de tipo `os.stat_result`.

```
import os

info_archivo = os.stat('arquivo.txt')

print(info_arquivo)

tamaño_arquivo = info_archivo.st_size
print(f"Tamaño do arquivo: {tamaño_archivo} bytes")

permisos_arquivo = info_arquivo.st_mode
print(f"Permisos do arquivo en octal: {oct(permisos_archivo)}")
```

O obxecto `os.stat_result` conta cas seguintes propiedades.

- `st_mode`: Permisos del arquivo ou directorio.
- `st_uid`: O UID do usuario propietario do arquivo ou directorio.
- `st_gid`: O GID do grupo propietario do arquivo ou directorio.
- `st_size`: O tamaño en bytes do arquivo.
- `st_atime`: O tempo de último acceso del arquivo (en timestamp).
- `st_mtime`: O tempo de última modificación del arquivo (en timestamp).

1.6. TRABALLAR CON DIRECTORIOS E FICHEIROS

1.6.1. Cambiar permisos

A función `os.chmod(path, mode)` utilízase para cambiar os permisos de acceso dun arquivo ou un directorio en **sistemas Unix**. O primeiro argumento é a ruta do ficheiro ou directorio e o segundo son os permisos que se lle aplicarán. Pode especificarse en formato octal ou utilizando constantes do módulo `stat`.

```
import os

# Establecer permisos 744 para o arquivo
os.chmod("arquivo.txt", 0o744)
```

Neste exemplo establécense os permisos de arquivo 744.

Neste outro exemplo utilízanse constantes do módulo `stat`:

```
import os
import stat

# Establecer permisos de só lectura para o propietario e o grupo
os.chmod("arquivo.txt", stat.S_IRUSR | stat.S_IRGRP)

# Establecer permisos de lectura, escritura e execución para o propietario
os.chmod("arquivo.txt", stat.S_IRWXU)
```

Algunhas das constantes máis comúns do módulo `stat` son:

- `stat.S_IRUSR`: Permiso de lectura para o propietario do arquivo.
- `stat.S_IWUSR`: Permiso de escritura para o propietario do arquivo.
- `stat.S_IXUSR`: Permiso de execución para o propietario do arquivo.
- `stat.S_IRGRP`: Permiso de lectura para o grupo do arquivo.
- `stat.S_IWGRP`: Permiso de escritura para o grupo do arquivo.
- `stat.S_IXGRP`: Permiso de execución para o grupo do arquivo.
- `stat.S_IROTH`: Permiso de lectura para outros usuarios.
- `stat.S_IWOTH`: Permiso de escritura para outros usuarios.
- `stat.S_IXOTH`: Permiso de execución para outros usuarios.

1.6.2. Cambiar propietario

A función `os.chown(path, uid, gid)` utilízase para **cambiar o propietario e o grupo dun arquivo ou directorio en sistemas tipo Unix**.

Os argumentos que reciben son:

- `path`: A ruta do arquivo ou directorio.
- `uid`: O identificador numérico do novo propietario.
- `gid`: O identificador numérico do novo grupo.

```
import os

# Cambiar O propietario e o grupo do arquivo
os.chown("arquivo.txt", 1000, 1000)
```

1.6.3. Cambiar de nome

A función `os.rename(src, dst)` utilízase para cambiar o nome dun arquivo ou directorio. Se non existe o directorio prodúcese un erro.

```
import os

# Cambiar o nome do arquivo "antigo_nome.txt" a "novo_nome.txt"
os.rename("antigo_nome.txt", "novo_nome.txt")
```

1.7. TRABALLAR CON DIRECTORIOS

1.7.1. Cambiar directorio de traballo

A función `os.chdir(ruta)` cambia o directorio de traballo do *script*. se a ruta non existe ou non ten permisos produce unha excepción. Esta función é equivalente a `cd` en Bash e `Set-Location` en PowerShell.

Por outro lado, a función `os.getcwd()` devolve o directorio de traballo do *script*. Esta función é equivalente a `pwd` en Bash e `Get-Location` en PowerShell.

```
import os

# Cambiar o directorio de traballo a "directorio"
os.chdir("directorio")

# Verificar o cambio de directorio de traballo
print("Directorio actual:", os.getcwd())
```

1.7.2. Listar directorio

`os.listdir(ruta)` utilízase para obter unha lista de nomes de arquivos e directorios dunha ruta especificada. Se non se proporciona ningunha ruta, devolve a lista de arquivos do directorio actual.

```
import os

# Obter unha lista de nomes de arquivos e directorios no directorio actual
arquivos = os.listdir()
print("Arquivos no directorio actual:", arquivos)

# Obtener unha lista de nomes de arquivos e directorios nunha ruta específica
ruta = os.path.join("ruta","de","proba")
print("Arquivos noutro directorio:", ruta)
```

Podemos combinar esta función con `os.path.isfile(path)` e `os.path.isdir(path)` para diferenciar entre directorios e ficheiros.

Esta función é similar en Bash a `ls` e en PowerShell a `Get-ChildItem` e os seus alias `dir` e `ls`.

1.7.3. Crear directorio

A función `os.mkdir(path)` utilízase para crear un novo directorio. O argumento que recibe é a ruta do directorio a crear.

```
import os

# Crea un novo directorio chamado "novo_directorio"
os.mkdir("novo_directorio")
```

A función `os.makedirs(ruta)` utilízase para crear un directorio e todos os seus directorio pais se non existen. Similar a `mkdir -p ruta` en Bash.

```
import os

# Crear un novo directorio xunto cos directorios parentais
```

```
os.makedirs("novo_directorio/novo_subdirectorio")
```

1.7.4. Eliminar directorio

A función `os.rmdir()` utilízase para eliminar un directorio **baleiro**. Se non está baleiro prodúcese unha excepción.

```
import os

# Elimina o directorio "directorio_baleiro"
os.rmdir("directorio_baleiro")
```

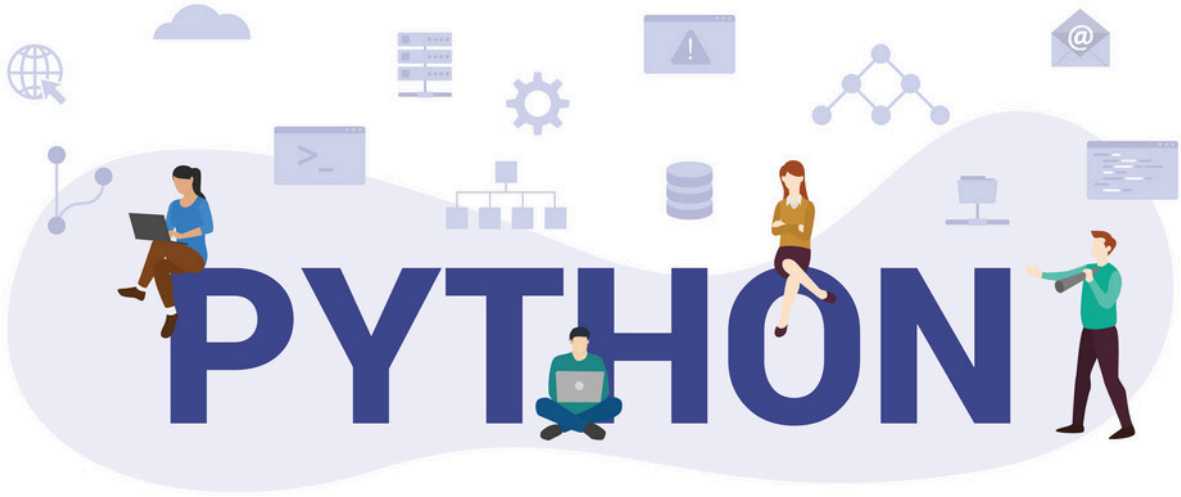
1.8. TRABALLAR CON FICHEIROS

1.8.1. Eliminar ficheiros

A función `os.remove(path)` elimina un ficheiro do sistema de ficheiros. Se non existe o ficheiro prodúcese unha excepción.

```
import os

# Eliminar o arquivo "arquivo.txt"
os.remove("arquivo.txt")
```

2. BIBLIOTECA SHUTIL



O módulo `shutil` proporciona unha gran variedade de funcións útiles para realizar operacións de alto nivel en arquivos e directorios.

2.1. REALIZAR COPIAS

A función `copy(orixe, destino)` utilízase para copiar un arquivo dende a localización de orixe a o ubicación destino. Se o ficheiro destino existe será substituído. localización Esta copia o contido do arquivo, pero non copia nin os permisos de arquivo nin os tempos de modificación. Se queremos manter estes metadatos debemos de utilizar a función `copy2(orixe, destino)`.

```
import shutil

src_file = 'arquivo_orixe.txt'
dst_file = 'arquivo-destino.txt'

shutil.copy(src_file, dst_file)
```

2.1.1. Copia recursiva

A función `shutil.copytree(src, dst)` utilizase para copiar todo unha árbore de directorios, incluídos tódolos arquivos e subdirectorios. É moi útil cando se quere copiar un directorio completo xunto o seu contido.

```
import shutil

src_dir = 'directorio_orixe'
dst_dir = 'directorio_destino'

shutil.copytree(src_dir, dst_dir)
```

2.2. ELIMINAR

A función `shutil.rmtree(directorio)` utilízase para eliminar un directorio e todo o seu contido de xeito recursivo. É dicir, elimina o directorio especificado e tódolos arquivos e directorios que contén.

```
import shutil

directorio_a_eliminar = 'directorio_a_eliminar'

shutil.rmtree(directorio_a_eliminar)
```

2.3. MOVER

A función `shutil.move(src, dst)` utilízase para mover arquivos ou directorios dunha localización noutra.

```
import shutil
arquivo_a_mover = 'arquivo_a_mover.txt'
nova_ubicacion = 'nova_ubicacion/arquivo_a_mover.txt'
shutil.move(arquivo_a_mover, nova_ubicacion)
```

2.4. COMPRESIÓN

2.4.1. Comprimir

A función `shutil.make_archive(nome_comprimido, formato, dir_a_comprimir)` utilízase para crear un arquivo comprimido que contén todo o contido dun directorio, incluídos os seus subdirectorios e arquivos.

- `nome_comprimido`: o nome base do ficheiro comprimido (obviando a extensión).
- `formato`. o formato do arquivo. Estas son algunhas opcións: 'zip', 'tar', 'gztar', 'bztar', 'xztar'.
- `dir_a_comprimir`. Directorio que se incluirá no ficheiro comprimido. Se non se indica comprímese o directorio actual.

```
import shutil

directorio_a_comprimir = 'directorio_a_comprimir'

shutil.make_archive('directorio_comprimido', 'zip', directorio_a_comprimir)
```

2.4.2. Descomprimir

Para descomprimir utilizamos a función `shutil.unpack_archive(arquivo_comprimido, directorio_saida)`. O directorio de saída é opcional, pódese omitir. Se se omiten os ficheiros descomprimiranse no directorio actual.

```
import shutil

arquivo_comprimido = 'arquivo_comprimido.zip'

directorio_destino = 'directorio_destino'

shutil.unpack_archive(arquivo_comprimido, extract_dir=directorio_destino)
```

3. BIBLIOTECA SUBPROCESS

Pódense executar comandos de Bash e PowerShell utilizando a biblioteca `subprocess`. Esta biblioteca proporciona un xeito de crear e xestionar procesos secundarios dende un *script* de Python.

3.1. FUNCIÓN RUN()

Coa función `run` podemos executar procesos:

```
import subprocess

# Executar un comando de Bash
resultado = subprocess.run(["ls", "-l"], capture_output=True, text=True)

# Imprimir a saida do comando
print("Saída do comando:")
print(resultado.stdout)
```

Este código executa o comando `ls -l`. Como vemos, a entrada do comando e unha lista cás diferentes partes do comando (o comando propiamente dito e as súas opcións). Por último imprime por consola.

O exemplo anterior executouse o comando en Bash. Podemos facelo mesmo en PowerShell:

```
import subprocess

# Executar un comando de PowerShell
resultado = subprocess.run(["powershell", "-Command", "Get-ChildItem"], capture_output=True,
text=True)

# Imprimir a saida do comando
print("Saida del comando:")
print(resultado.stdout)
```

O argumento `capture_output` utilízase para indicar se se debe capturar a saída estándar e de error do proceso secundario. Deste xeito podemos acceder a saída estándar no atributo `stdout` e a saída de erro no atributo `stderr`. Se se establece a `False` non se captura dita saída.

O argumento `text=True` utilízase para especificar que a entrada e saída do proceso secundario debe ser en formato texto en lugar de formato de bytes.

No seguinte exemplo proporcionamos unha función que obtén unha lista de arquivos en calquera dos dous sistemas operativos:

```
import subprocess
import os

def ls():
    if os.name == 'nt':  # Windows
        comando = ['dir', '/B']
    elif os.name == 'posix':  # Linux o macOS
        comando = ['ls']
    else:
        print("Sistema operativo non compatible.")
        return None

    try:
        resultado = subprocess.run(comando, capture_output=True, text=True)
        lista = resultado.stdout.splitlines()
        return lista
    except subprocess.CalledProcessError as e:
        print("Erro ao executar o comando:", e)
        return None
```

3.2. OBXECTO POPEN

Tamén se pode utilizar a función `subprocess.Popen()` para executar comandos. Este proporciona maior control sobre o proceso. Este e especialmente útil se é necesario interactuar co proceso ou se este se necesita executar en segundo plano.

```
import subprocess

# Comando a executar
comando = ["ls", "-l"]

# Crea un obxecto Popen para executar o comando
proceso = subprocess.Popen(comando, stdout=subprocess.PIPE, stderr=subprocess.PIPE,
text=True)

# Obtense a saida e erros do proceso
saida, erros = proceso.communicate()

# Imprimir la salida y errores
print("Saida do comando:")
print(saida)

print("\nErros do comando:")
print(erros)
```

Os argumentos `stdout=subprocess.PIPE` y `stderr=subprocess.PIPE` utilízanse para obter a saída e os erros dos procesos.

3.2.1. Interactividade co comando

A función `subprocess.Popen()` xunto coa redirección da entrada/saída estándar do proceso para enviar respostas dende un *script* de Python.

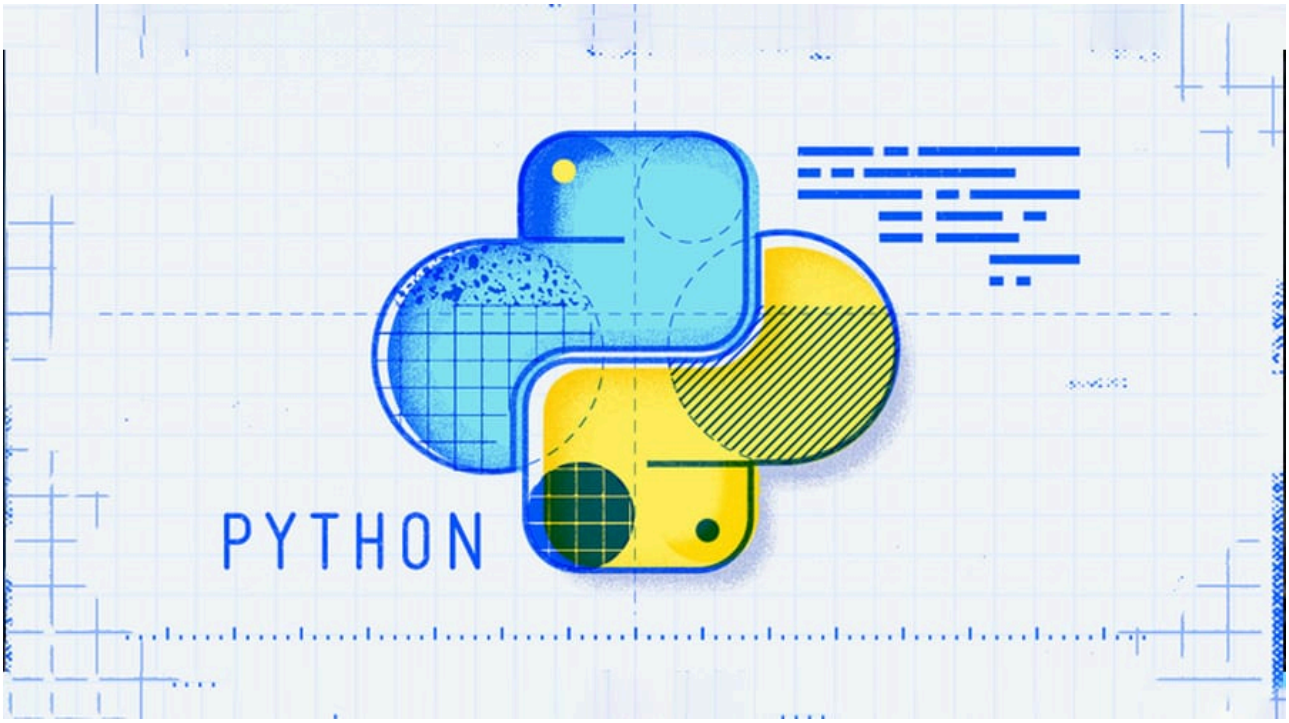
Aquí un exemplo de como agregar un usuario nun sistema Linux utilizando o comando `adduser`:

```
import subprocess

# Solicitar ao usuario o nome do novo usuario
novo_usuario = input("Ingrese o nome do novo usuario: ")

# Comando para agregar un usuario
comando = ["sudo", "adduser", novo_usuario]

# Crea un obxecto Popen para executar o comando
proceso = subprocess.Popen(comando, stdin=subprocess.PIPE, stdout=subprocess.PIPE,
stderr=subprocess.PIPE, text=True)

# Pide ao usuario que ingrese o contrasinal para o novo usuario
contrasinal = input(f"Ingrese o contrasinal para o usuario {novo_usuario}: ")

# Enviamos o contrasinal ao proceso
proceso.stdin.write(novo_usuario + '\n')
# Aseguramos de que a entrada se envía correctamente
proceso.stdin.flush()
# Cerra a entrada estándar do proceso
proceso.stdin.close()

# Obtén a salída e os erros do proceso
saida, erros = proceso.communicate()

# Imprimir la salida y errores
print("Salida del proceso:")
print(saida)

print("\nErrores del proceso:")
```

```
print(erros)
```

---

4. BIBLIOTECA PWD



A librería `pwd` **é específica de sistemas Unix** e non está dispoñible en Windoes. A función principal é proporcionar acceso a base de datos de contrasinais do sistema, permitindo obter información sobre os usuarios do sistema como o seu nome, UID, etc.

4.1. ESTRUTURA DE PWD

A estrutura `struct_passwd` é un obxecto que contén información sobre un usuario do sistema operativo nun sistema Unix.

Esta estrutura é devolta polas funcións da libraría `pwd`, como `getpwuid()` ou `getpwnam()`, e contén os seguintes campos:

- `pw_name`: O nome do usuario.
- `pw_passwd`: O contrasinal do usuario (encriptado).
- `pw_uid`: O identificador de usuario (UID).
- `pw_gid`: O identificador de grupo (GID) primario do usuario.
- `pw_gecos`: Información adicional do usuario (por exemplo, o nome completo).
- `pw_dir`: O directorio de inicio do usuario.
- `pw_shell`: A shell predeterminada do usuario.

4.2. FUNCIÓNS

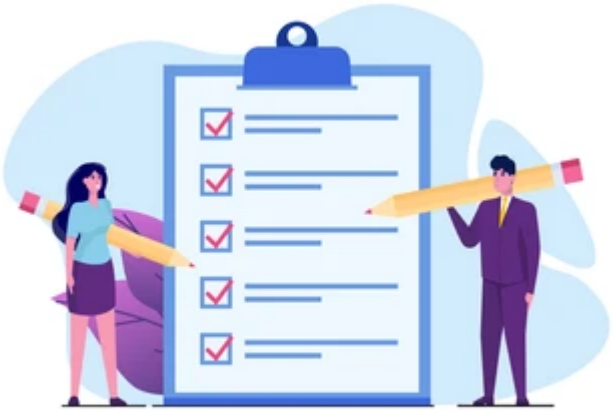
A libraría `pwd` en Python proporciona funcións para traballar con información de usuarios do sistema operativo nun entorno Unix.

A función **`getpwuid(uid)`** devolve un obxecto `pwd.struct_passwd` que contén información dun usuario dado o seu identificador de usuario (`uid`). A través deste obxecto, podes acceder a detalles como o nome do usuario, o seu directorio de inicio, a shell predeterminada, entre outros.

A función **`getpwnam(name)`** devolve un obxecto `pwd.struct_passwd` que contén información dun usuario dado o seu nome de usuario (`name`). Utiliza o nome de usuario para atopar a información asociada a ese usuario.

```
import pwd

def obter_info_usuario(uid):
    info_usuario = pwd.getpwuid(uid)

    # Imprime a información do usuario
    print("Nome do usuario:", info_usuario.pw_name)
    print("Directorio de inicio:", info_usuario.pw_dir)
    print("Shell predeterminada:", info_usuario.pw_shell)

# Chama á función e pasa o UID do usuario para obter a súa información
obter_info_usuario(1000)
```

5. EXERCICIOS



5.1. ENUNCIADOS

1. Escribe unha función que conte a cantidade de ficheiros dun directorio dado por argumento. Deberá devolver a cantidade.
2. Escribe unha función que busque os arquivos con unha extensión específica nun directorio. Recibirá como parámetro obrigatorio a extensión e como opcional a ruta dun directorio. Se non se pasa este dato utilizarase o directorio actual. Devolve unha lista de nomes de ficheiros.
3. Modifica o exercicio anterior para que ademais se realice unha busca nos subdirectorios do directorio.
4. Escribe unha función que elimine os arquivos dun directorio se teñen máis dunha cantidade de días. Recibirá como parámetro obrigatorio a cantidade de días como un enteiro e como opcional a ruta dun directorio. Se non se pasa este dato utilizarase o directorio actual.
5. Escribe unha función no que a tódolos ficheiros se lles engada a extensión “backup” dun directorio dado por parámetros.
6. Escribe unha función que devolva o ficheiro máis grande dentro do directorio actual e os seus subdirectorios. Débese de utilizar recursividade para simplificar dito algoritmo. A función devolverá a ruta absoluta do ficheiro.
7. Escribe unha función que reciba a ruta dun directorio e unha cadea de texto. Esta debe buscar en que ficheiros “.txt” se encontra dito texto como contido. Debe devolver unha lista con tódolos nomes deses ficheiros.
8. Escribe unha función que substitúa unha palabra dun ficheiro de texto por outra. A función recibirá a ruta do ficheiro, a palabra a substituír e o texto co se realizará o cambio.
9. Crea un *script* que a partir dun ficheiro CSV cree usuarios nun sistema Linux. O nome de usuario será a primeira letra do seu nome seguido do apelido, todo en minúscula. Se dous usuarios tiveran o mesmo nome de usuario, engádelle ao final o número 1. Se volverá haber máis coincidencias, en lugar do 1 ponlle o número 2, e así sucesivamente. O ficheiro CSV ten o seguinte formato:

```
nome,apelido
Juan,Perez
Maria,Gonzalez
Carlos,Rodriguez
Jose,Perez
```

10. A partir do ficheiro CSV anterior, crea os directorios “home” de cada usuario antes de engadilos usuarios. Asígnalle dito directorio na súa creación e por último cambia o usuario propietario de dito directorio.
11. Fai o exercicio anterior, pero en lugar de obtela información dun ficheiro CSV faino a partir deste ficheiro JSON.

```
[
    {"nome": "Juan", "apelido": "Perez"},
    {"nome": "Maria", "apelido": "Gonzalez"},
    {"nome": "Carlos", "apelido": "Rodriguez"},
    {"nome": "Jose", "apelido": "Perez"}
]
```

12. Crea un *script* que copie tódolos ficheiros do sistema coa extensión “py” no directorio “python”. Debes comprobar se existe este directorio, e se existe borrar todo o seu contido. Fai que o *script* se poida executar en Windows e en Linux. Podes axudarte desta función para obter tódalas unidades en Windows.

```
def obtener_unidades_windows():
    unidades = []
    # Recorremos as letras de unidades da A a Z
    for letra in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
        unidade = f"{letra}:\\"
        # Verificamos se a unidade existe como directorio
        if os.path.isdir(unidade):
            unidades.append(unidade)
    return unidades
```

13. Crea unha función en Python para realizar unha copia de seguridade dun directorio. Débese gardar cada copia de seguridade nun arquivo comprimido que terá como nome o día, mes e ano do momento no que se execute. A función recibirá a carpeta da que se fará a copia de seguridade e o directorio donde se almacenará dita copia.

14. A partir do ficheiro `/etc/fstab` e utilizando expresións regulares, crea unha función en Python no que se obteñan tódolos dispositivos montados e o seu punto de montaxe a través deste ficheiro. Devolve esta información nunha lista de dicionarios.

15. Fai un *script* con dúas funcións que permitirá eliminar arquivos duplicados nun directorio:

- `obter_hash(directorio)`: a partir dun directorio que recibe como argumento, devolve un dicionario onde cada clave e o *hash* do contido dos ficheiros do directorio e o seu valor é unha lista coa ruta absoluta de cada un dos ficheiros que contén con ese *hash*. No seguinte exemplo explícase como obter o *hash* dun ficheiro.

```
import hashlib

# rb permite ler un ficheiro en binario
with open(ruta_absoluta, 'rb') as archivo:
    contenido = archivo.read()
    # isto devolve o hash do ficheiro
    hash_archivo = hashlib.sha256(contenido).hexdigest()
```

- `obter_hash(directorio)`: a partir da información do dicionario anterior elimina os arquivos duplicados. Almacena nun ficheiro `eliminados.log` tódolos arquivos eliminados.

16. Fai unha función en Python que instale un paquete utilizando `apt-get` sen utilizar a opción `-y`. Antes da instalación executa `apt-get update` O nome do paquete debe de pasárselle como argumento a función.
