



GET YOUR SYSTEM INFORMATION - USING PYTHON SCRIPT

Document Date: September 2023 || Document: #369

Author: Fatemeh Nabidoust

Email: nabidoustf@gmail.com | **Web:** <https://aeip.ir/>



INTRODUCTION: SYSTEM INFORMATION USING PYTHON

In today's fast-paced technological world, having the ability to gather system information is crucial for a variety of purposes, from software development and troubleshooting to system monitoring and optimization. Python, with its simplicity and versatility, provides a powerful way to access and extract system information

In this mini article, we will explore how to gather platform system information using Python, covering various aspects from basic to advanced techniques.

Python offers several libraries and modules that allow us to access and retrieve various types of system information. Whether you're a developer aiming to optimize your software for different platforms or a system administrator monitoring server health, having the ability to extract key system details is invaluable.



Your System Information - Using Python Script

Getting system information for your system can easily be done by the operating system in use. we will look into various ways to derive your system information using Python. There are two ways to get information:

- Using Platform module
- subprocess

It's also useful to use third-party libraries that make your job really easier, instead of reinventing the wheel every time. In this tutorial you will learn with psutil which is a cross-platform library for process and system monitoring in Python. There are quite popular tools to extract system and hardware information in Linux, such as lshw, uname and hostnamectl. However, we'll be using the psutil library in Python so it can run on all operating systems and get almost identical results.

TABLE OF CONTENT OF THIS TUTORIAL LEVEL 1

1. Using Platform module:

Installation of the platform module can be done using the below command:

- pip install platform

To start off, let's learn how to obtain some basic computer system information using Python. The platform module is a built-in package that provides cross-platform access to system-related details:

```
1 import platform
2 system_info=platform.uname()
3 print(f"System: {system_info.system}")
4 print(f"Node Name: {system_info.node}")
5 print(f"Release: {system_info.release}")
6 print(f"Version: {system_info.version}")
7 print(f"Machine: {system_info.machine}")
8 print(f"Processor: {system_info.processor}")
```

Using WMI module (only for Windows):

The WMI module can be used to gain system information of a windows machine and can be installed using the below command:

- pip install wmi

```
1 import wmi
2 c=wmi.WMI()
3 system_info=c.Win32_ComputerSystem()[0]
4 print(f"Manufacturer: {system_info.Manufacturer}")
5 print(f"Model: {system_info.Model}")
6 print(f"Name: {system_info.Name}")
7 print(f"NumberOfProcessors: {system_info.NumberOfProcessors}")
8 print(f"SystemType: {system_info.SystemType}")
9 print(f"SystemFamily: {system_info.SystemFamily}")
```

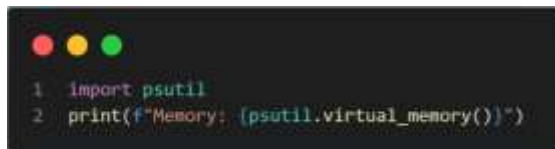
Using OS module (only for Unix):

```
1 import os
2 print(os.uname())
```

Using psutil module:

used for getting runtime process information on the system.

- pip install psutil



```
1 import psutil
2 print(f"Memory: {psutil.virtual_memory()}")
```

USING THE SUBPROCESS MODULE LEVEL 2

subprocess module

the subprocess module use to interact with cmd and to retrieve information into python.

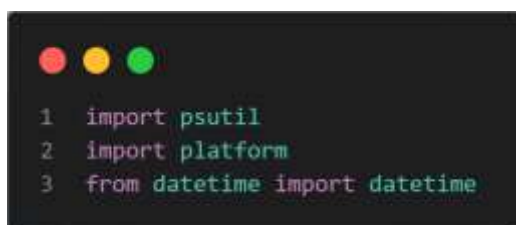


```
1 import subprocess
2
3 # traverse the info
4 Id = subprocess.check_output(['systeminfo']).decode('utf-8').split('\n')
5 new = []
6
7 # arrange the string into clear info
8 for item in Id:
9     new.append(str(item.split("\r")[:-1]))
10 for i in new:
11     print(i[2:-2])
```

BASIC SYSTEM INFORMATION LEVEL 3

Basic System Information

Open up a new Python file and let's get started, importing the necessary modules:



```
1 import psutil
2 import platform
3 from datetime import datetime
```

CPU Information

Let's make a function that converts a large number of bytes into a scaled format (e.g in kilo, mega, Giga, etc.):

```
1 def get_size(bytes, suffix="B"):
2     """
3     Scale bytes to its proper format
4     e.g:
5         1253656 => '1.28MB'
6         1253656678 => '1.17GB'
7     """
8     factor = 1024
9     for unit in ["", "K", "M", "G", "T", "P"]:
10        if bytes < factor:
11            return f"{bytes:.2f}{unit}{suffix}"
12        bytes /= factor
```

We gonna need the platform module here:

```
1 print("="*40, "System Information", "="*40)
2 uname = platform.uname()
3 print(f"System: {uname.system}")
4 print(f"Node Name: {uname.node}")
5 print(f"Release: {uname.release}")
6 print(f"Version: {uname.version}")
7 print(f"Machine: {uname.machine}")
8 print(f"Processor: {uname.processor}")
```

Getting the date and time the computer was booted:

```
1 # Boot Time
2 print("="*40, "Boot Time", "="*40)
3 boot_time_timestamp = psutil.boot_time()
4 bt = datetime.fromtimestamp(boot_time_timestamp)
5 print(f"Boot Time: {bt.year}/{bt.month}/{bt.day} {bt.hour}:{bt.minute}:{bt.second}")
```

Let's get some CPU information, such as the total number of cores, usage, etc

```

1 # let's print CPU information
2 print("="*40, "CPU Info", "="*40)
3 # number of cores
4 print("Physical cores:", psutil.cpu_count(logical=False))
5 print("Total cores:", psutil.cpu_count(logical=True))
6 # CPU frequencies
7 cpufreq = psutil.cpu_freq()
8 print(f"Max Frequency: {cpufreq.max:.2f}Mhz")
9 print(f"Min Frequency: {cpufreq.min:.2f}Mhz")
10 print(f"Current Frequency: {cpufreq.current:.2f}Mhz")
11 # CPU usage
12 print("CPU Usage Per Core:")
13 for i, percentage in enumerate(psutil.cpu_percent(percpu=True, interval=1)):
14     print(f"Core {i}: {percentage}%")
15 print(f"Total CPU Usage: {psutil.cpu_percent()}%")

```

psutil's `cpu_count()` function returns number of cores, whereas `cpu_freq()` function returns CPU frequency as a namedtuple including current, min, and max frequency expressed in Mhz, you can set `percpu=True` to get per CPU frequency.

`cpu_percent()` method returns a float representing the current CPU utilization as a percentage, setting interval to 1 (seconds) will compare system CPU times elapsed before and after a second, we set `percpu` to True in order to get CPU usage of each core.

MEMORY Usage

```

1 # Memory Information
2 print("="*40, "Memory Information", "="*40)
3 # get the memory details
4 svmem = psutil.virtual_memory()
5 print(f"Total: {get_size(svmem.total)}")
6 print(f"Available: {get_size(svmem.available)}")
7 print(f"Used: {get_size(svmem.used)}")
8 print(f"Percentage: {svmem.percent}%")
9 print("="*20, "SWAP", "="*20)
10 # get the swap memory details (if exists)
11 swap = psutil.swap_memory()
12 print(f"Total: {get_size(swap.total)}")
13 print(f"Free: {get_size(swap.free)}")
14 print(f"Used: {get_size(swap.used)}")
15 print(f"Percentage: {swap.percent}%")

```

virtual_memory() method returns stats about system memory usage as a namedtuple, including fields such as total (total physical memory available), available (available memory, i.e not used), used and percent (i.e percentage). swap_memory() is the same but for swap memory.

We used the previously defined get_size() function to print values in a scaled manner, as these statistics are expressed in bytes.

Disk Usage

```

1 # Disk Information
2 print("="*40, "Disk Information", "="*40)
3 print("Partitions and Usage:")
4 # get all disk partitions
5 partitions = psutil.disk_partitions()
6 for partition in partitions:
7     print(f"=== Device: {partition.device} ===")
8     print(f" Mountpoint: {partition.mountpoint}")
9     print(f" File system type: {partition.fstype}")
10    try:
11        partition_usage = psutil.disk_usage(partition.mountpoint)
12    except PermissionError:
13        # this can be caught due to the disk that
14        # isn't ready
15        continue
16    print(f" Total Size: {get_size(partition_usage.total)}")
17    print(f" Used: {get_size(partition_usage.used)}")
18    print(f" Free: {get_size(partition_usage.free)}")
19    print(f" Percentage: {partition_usage.percent}%")
20 # get IO statistics since boot
21 disk_io = psutil.disk_io_counters()
22 print(f"Total read: {get_size(disk_io.read_bytes)}")
23 print(f"Total write: {get_size(disk_io.write_bytes)}")

```

As expected, disk_usage() function return disk usage statistics as a namedtuple, including total, used and free space expressed in bytes.

Network Information

```

1 # Network information
2 print("="*40, "Network Information", "="*40)
3 # get all network interfaces (virtual and physical)
4 if_addrs = psutil.net_if_addrs()
5 for interface_name, interface_addresses in if_addrs.items():
6     for address in interface_addresses:
7         print(f"=== Interface: {interface_name} ===")
8         if str(address.family) == 'AddressFamily.AF_INET':
9             print(f"  IP Address: {address.address}")
10            print(f"  Netmask: {address.netmask}")
11            print(f"  Broadcast IP: {address.broadcast}")
12        elif str(address.family) == 'AddressFamily.AF_PACKET':
13            print(f"  MAC Address: {address.address}")
14            print(f"  Netmask: {address.netmask}")
15            print(f"  Broadcast MAC: {address.broadcast}")
16 # get IO statistics since boot
17 net_io = psutil.net_io_counters()
18 print(f"Total Bytes Sent: {get_size(net_io.bytes_sent)}")
19 print(f"Total Bytes Received: {get_size(net_io.bytes_recv)}")

```

The `net_if_addrs()` function returns the addresses associated with each network interface card installed on the system. For extracting detailed network usage, check this tutorial that's dedicated to network usage monitoring with `psutil`.

GPU Information

`psutil` doesn't provide us with GPU information. Therefore, we need to install `GPUtil`:

- `pip3 install gputil`

`GPUtil` is a Python module for getting the GPU status for NVIDIA GPUs only, it locates all GPUs on the computer, determines their availability, and returns an ordered list of available GPUs. It requires the latest NVIDIA driver installed.

Also, we need to install `tabulate` module, which will allow us to print GPU information in a tabular way:

- `pip3 install tabulate`

The following lines of code print all GPUs in your machine along with their details:


```

1 import GPUtil
2 from tabulate import tabulate
3 print("="*40, "GPU Details", "="*40)
4 gpus = GPUtil.getGPUs()
5 list_gpus = []
6 for gpu in gpus:
7     # get the GPU id
8     gpu_id = gpu.id
9     # name of GPU
10    gpu_name = gpu.name
11    # get % percentage of GPU usage of that GPU
12    gpu_load = f"{gpu.load*100}%"
13    # get free memory in MB format
14    gpu_free_memory = f"{gpu.memoryFree}MB"
15    # get used memory
16    gpu_used_memory = f"{gpu.memoryUsed}MB"
17    # get total memory
18    gpu_total_memory = f"{gpu.memoryTotal}MB"
19    # get GPU temperature in Celsius
20    gpu_temperature = f"{gpu.temperature} °C"
21    gpu_uuid = gpu.uuid
22    list_gpus.append((
23        gpu_id, gpu_name, gpu_load, gpu_free_memory, gpu_used_memory,
24        gpu_total_memory, gpu_temperature, gpu_uuid
25    ))
26
27 print(tabulate(list_gpus, headers=('id', 'name', 'load', 'free memory', 'used memory', 'total memory',
28                                'temperature', 'uuid')))

```

Conclusion

In this mini article, we've explored how to gather platform system information using Python. From basic details like system name and architecture to advanced metrics like CPU usage and memory statistics, Python's libraries empower developers and administrators to effectively monitor, optimize, and troubleshoot systems. Whether you're building cross-platform applications or ensuring server health, the ability to access system information is an essential skill in the modern tech landscape.