1. Python Common Things
   1. Life cycle of process
   2. Life-cycle of instance object
   3. 10 elements for python function definition
2. Python VS Node js
3. Syllabus
   1. Basics
   2. Intermediate
   3. Libraries
   4. Frameworks
   5. Advance
   6. Web Development
   7. DS
4. Data Model
5. Python Version important points
6. Interview Question Set
   1. Beginner
   2. Intermediate
   3. Advance
   4. All levels
   5. Code practice questions
7. Online
   1. Code practice tools
   2. Study portal website

**Python Common Things**

**Life-cycle of Python process**

Python is not inherently single-threaded. It has full support for multi-threading, meaning that you can have multiple threads running concurrently within a single process. However, the most commonly used implementation of Python, CPython, has a Global Interpreter Lock (GIL) that allows only one thread to execute Python code at a time. This means that even though you can have multiple threads in a Python program, only one thread can be executing Python code at any given time. Other implementations of Python, such as Jython, IronPython, and PyPy, do not have a GIL and can run multiple threads in parallel.

So while Python as a language is not single-threaded, the most commonly used implementation of it has limitations that can make it behave like it is single-threaded in certain situations. However, there are still many use cases where multi-threading in Python can be beneficial, such as when performing I/O-bound tasks or when using certain libraries that release the GIL. It’s also possible to achieve true parallelism in Python by using multiple processes instead of threads.

A Python process may progress through three steps of its life-cycle: a new process, a running process, and a terminated process.

While running, the process may be executing code or may be blocked, waiting on something such as another process or an external resource. Although not all processes may block, it is optional based on the specific use case for the new process.

1. New Process.
2. Running Process.
   1. Blocked Process (optional).
3. Terminated Process.

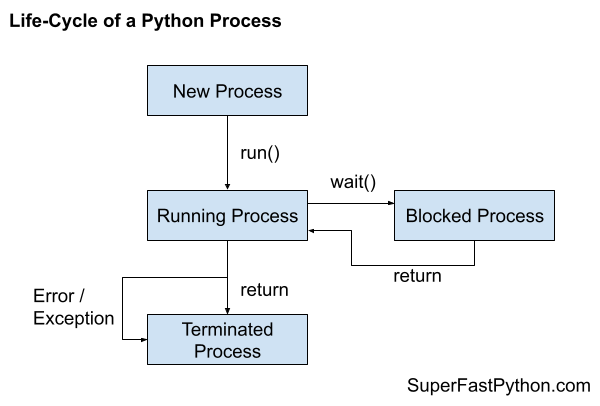
A new process is a process that has been constructed by creating an instance of the multiprocessing.Process class.

A new process can transition to a running process by calling the start() function.

A running process may block in many ways, such as reading or writing from a file or a socket or by waiting on a synchronization primitive such as a semaphore or a lock. After blocking, the process will run again.

Finally, a process may terminate once it has finished executing its code or by raising an error or exception.

The following figure summarizes the states of the process life-cycle and how the process may transition through these states.



*https://superfastpython.com/process-life-cycle/*

1.1 Start Does Not Block

The start() function does not block.

This means that it returns immediately. It does not wait and return after the new process has terminated.

If you need to wait until the new process finishes, you can join the new process via the join() function.

1.2 Start Does Not Take Arguments

The start() function and the internal run() function do not take arguments.

If you need to pass arguments to your new process, you can via the “args” argument of the constructor of the multiprocessing.Process class or as arguments to the constructor of your overridden process class.

1.3 Start Does Not Return Values

The start() function and the internal run() function do not return values.

If you need to return values from your new child process to the caller parent process or another process, you have a number of options. Such as:

* Share data using a shared ctype.
* Share data between processes using a pipe.
* Share data between processes using a queue.

Step 2a: Blocked Process

A running process may make a call to a function that blocks.

In concurrency programming, a blocking function call means a function call that waits on some event or condition.

This may involve waiting for another process via concurrency primitive, such as:

* Waiting for a mutex lock.
* Waiting on a barrier.
* Waiting on a semaphore.
* Waiting for an event.
* Waiting for a process to terminate.

And so on.

It may also involve waiting for a blocking IO, such as reading or writing from an external device, such as:

* A file on the hard drive.
* A socket on a local or remote server.
* A device like a printer, external drive, peripheral, or screen.

And so on.

Step 3: Terminated Process

A process is terminated after the run() function returns or exits.

The run() function may exit normally if your custom function finishes normally, if using the “target” keyword on the multiprocessing.Process constructor. It may also exit normally if you override the run() function when extending the multiprocessing.Process class.

Exiting normally means that the end of the function was reached and the function returned. It may also mean that you returned from the function directly with the “return” statement.

* Restart a Process in Python
* Kill a Process in Python
* Safely Stop a Process in Python

Some other topics:

* Effective Python
* High Performance Python
* Python in a Nutshell
* Python Multiprocessing: The Complete Guide
* multiprocessing — Process-based parallelism
* Addition of the multiprocessing package

**The Life Cycle of Python Instance Objects**

*https://betterprogramming.pub/the-life-cycle-of-python-instance-objects-4a719fb4e925*

**Part I. Key Terminology**

Instance and Instantiation

An instance is also known as an instance object, which is the actual object of the class that holds the data.

The process of creating an instance is known as instantiation which is allocation of memory to the object

Initialization and Constructor

The constructor is the method that you use to create your custom class. Suppose that the custom class Person defines the \_\_init\_\_ method as the following signature: \_\_init\_\_(self, name), which is essentially equivalent to the constructor method Person(name). execution of which is by default through calling the \_\_init\_\_ method.

Variable and Object

In Python, objects are actual data items stored in the memory, while variables are labels that refer to the underlying objects. In other words, we—programmers — play with variables, while Python, the interpreter handles the objects based on our interaction with variables

Namespaces and Scopes

When Python tries to interpret your code, one key task is to understand what variables and functions that you refer to. Python uses namespaces to track the variables and functions that it can search.

Scopes define boundaries where Python can resolve the variables.

Reference Counting and Garbage Collection

Related to the previous concept, each variable holds a reference to the underlying object. Python tracks how many references that an underlying object has to decide if it needs to keep the objects alive or not in the memory.

[*https://stackify.com/python-garbage-collection/*](https://stackify.com/python-garbage-collection/)

In normal use cases, we don’t need to manage memory allocation and deallocation for Python objects because Python uses reference counting to free up the memory space. However, there are occasions when reference counting won’t work (e.g., reference cycles). In other words, there is garbage (e.g., objects actually not needed but have non-zero references) in the memory. In this case, Python will try freeing up the memory used by these objects using the garbage collection mechanisms.

De-initialization

The process of removing a particular object from the memory is called de-initialization. It can happen when the reference counting for an object reaches zero or execution of the entire program is completed. Some people also refer to this process as destruction.

**Part II. The Typical Life Cycle**

Instantiation

*# Create an empty list*

*a\_list = list() # [1, 2, 3]*

*# Create an empty dictionary*

*a\_dict = dict() # {“a”: 1, “b”: 2}*

*# Create an empty set*

*a\_set = set()*

*# Create an empty tuple*

*a\_tuple = tuple()*

*https://docs.python.org/3/reference/datamodel.html*

The instantiation involves two key steps.

1. Creation of the instance object in the memory. This step involves calling the \_\_new\_\_() method, the name of which simply means creating a new instance object. The key operation here is to allocate memory to the object that you’re constructing. After creating the object, you can see that a memory slot has been associated with the object by calling the id() function, which reveals the memory address of the underlying object with the default CPython implementation.
2. Set the initial state of the object. Once the object is created in the memory, Python will call the \_\_init\_\_() method to complete the initialization process for the object. If you compare the instance before and after setting the name attribute, you’ll notice that the new object didn’t have any attributes to start with, and after initialization, it has the expected name attribute. Notably, we’re still working on the same object that is created using the \_\_new\_\_() method, which has the same memory address.

Active in Namespaces

*>>> globals().keys()*

*dict\_keys(['\_\_name\_\_', '\_\_doc\_\_', '\_\_package\_\_',*

*'\_\_loader\_\_', '\_\_spec\_\_', '\_\_file\_\_',*

*'\_\_builtins\_\_', 'sys', 'Person', 'person'])*

*>>> globals()['person']*

*<\_\_main\_\_.Person object at 0x10c8719d0>*

*>>> id(globals()['person']) # 4505147856*

*>>> hex(id(person)) # '0x10c8719d0'*

When an object is active in the namespace, Python also tracks how many objects hold reference to the object for memory management purposes. To find out the reference count, we can take advantage of the functionalities provided by the sys and gc modules.

*>>> import sys*

*... import gc*

*...*

*>>> # With the sys module*

*>>> sys.getrefcount(person)*

*2*

*>>> # With the gc module*

*>>> len(gc.get\_referrers(person))*

*1*

The result that we get from the sys.getrefcount(person) call, because it also includes a temporary reference to the object of the function call itself, and this behavior is expected, as discussed by the official documentation

*https://docs.python.org/3/library/sys.html#sys.getrefcount*

*>>> # Create an object that references to the person variable*

*>>> family = {'household head': person}*

*>>> len(gc.get\_referrers(person)) 2*

*>>> # Delete the object that references to the person variable*

*>>> del family*

*>>> len(gc.get\_referrers(person)) 1*

Destruction

When the reference count of an object drops to zero, it will be removed from the memory, and this is the process of destructing the instance object. In normal cases, we don’t need to worry about these things, because Python manages memory for us automatically

*>>> 'person' in globals() True*

*>>> del person*

*>>> 'person' in globals() False*

Notably, deleting one of them doesn’t trigger the calling of the \_\_del\_\_ method that we implemented for the Person class. However, when the underlying object is about to be deleted because the object’s reference count is zero after deleting the second variable.

In other words, the del statement only removes the variable label from the corresponding namespace, while the \_\_del\_\_ method is concerned about the underlying object when it is to be destructed from the memory.

**10 Elements When You Define Python Functions**

1. Explicit and meaningful names

# A few explicit and meaningful names

def get\_account\_info(): pass

Small and Single Purpose

Don’t reinvent the wheel

For instance, if you work with data in the CSV format, you can look into the functionalities in the CSV module. Alternatively, the pandas library can handle CSV files gracefully. For another instance, if you want to count elements in a list, you should consider the Counter class in the collections module, which is designed specifically for these operations.

2. Default Arguments

Relevant scenarios

Set default arguments

*def set\_sale\_price(discount, additional\_discount=1):*

Avoid the pitfalls of mutable default arguments

*>>> def add\_item\_to\_cart(new\_item, shopper\_name, existing\_items=[]):*

*... existing\_items.append(new\_item)*

*... print(f"{shopper\_name}'s cart has {existing\_items}")*

*... return existing\_items*

*...*

*... shopping\_list\_wife = add\_item\_to\_cart("Dress", "Jennifer")*

*... shopping\_list\_husband = add\_item\_to\_cart("Soccer", "David")*

*...*

*Jennifer's cart has ['Dress']*

*David's cart has ['Dress', 'Soccer']*

3. Consider Returning Multiple Values

Multiple values in a tuple

*>>> from statistics import mean, stdev*

*... def evaluate\_test\_result(scores):*

*... scores\_mean = mean(scores)*

*... scores\_std = stdev(scores)*

*... return scores\_mean, scores\_std*

*...*

*... evaluation\_result = evaluate\_test\_result([1, 1, 1, 2, 2, 2, 6, 6, 6])*

*... print(f"Evaluation Result ({type(evaluation\_result)}): {evaluation\_result}")*

*...*

*Evaluation Result (<class 'tuple'>): (3, 2.29128784747792)*

But no more than three

It can create a lot of confusion for the users over which are which

4. Use Try…Except

not discussed here, it’s typically a good idea to define custom exceptions specific to your tool if you want to give more specific information.

*def get\_data\_from\_file(filepath):*

*try:*

*with open(filepath) as file:*

*computed\_value = process\_data(file)*

*except Exception:*

*raise SomeCustomFileException(f"can't open the file at the path: {filepath}")*

*return computed\_value*

5. Consider Argument Validation

*# Check type before running the code*

*def add\_numbers(a, b):*

*if not(isinstance(a, (float, int)) and isinstance(b, (float, int))):*

*raise TypeError("Numbers are required.")*

*return a + b*

Discussion: EAFP vs. LBYL

Although EAFP is a preferred coding style in the Python world, depending on your use case, you should also consider using LBYL which can provide more user-friendly function-specific error messages than the generic built-in error messages you get with the EAFP style.

6. Consider Lambda Functions As Alternatives

Functions as parameters of other functions

Lambda functions as alternatives

A lambda function is an anonymous function declared using the lambda keyword. It takes zero to more arguments and has one expression for applicable operations with the form: lambda arguments: expression.

*>>> sorted(grades, key=lambda x: x['score'])*

*>>> import pandas as pd*

*>>> interest\_rates = pd.Series([0.023, 0.025, 0.037])*

*>>> interest\_rates.map(lambda x: f"{x:.2%}")*

*0 2.30%*

*1 2.50%*

*2 3.70%*

*dtype: object*

7. Consider Decorators

Decorators are functions that modify the behavior of other functions without affecting their core functionalities. In other words, they provide modifications to the decorated functions at the cosmetic level.

Shown below, the decorator function simply runs the decorated function twice. To use the decorator, we simply place the decorator function name above the decorated function with an @ prefix.

*>>> # Define a decorator function*

*... def echo\_wrapper(func):*

*... def wrapper(\*args, \*\*kwargs):*

*... func(\*args, \*\*kwargs)*

*... func(\*args, \*\*kwargs)*

*... return wrapper*

*...*

*>>> # Define a function that is decorated by echo\_wrapper*

*... @echo\_wrapper*

*... def say\_hello():*

*... print('Hello!')*

*...*

*>>> # Call the decorated function*

*... say\_hello()*

*Hello!*

*Hello!*

Use decorators in function declarations

For instance, the @property decorator converts an instance method to make it behave like a regular attribute, which allows the access of using the dot notation.

*>>> class Product:*

*... def \_\_init\_\_(self, item\_id, price):*

*... self.item\_id = item\_id*

*... self.price = price*

*...*

*... @property*

*... def employee\_price(self):*

*... return self.price \* 0.9*

*...*

*>>> product = Product(12345, 100)*

*>>> product.employee\_price*

*90.0*

Another trivial use case of decorators is the time logging decorator, which can be particularly handy when the efficiency of your functions is of concern.

8. Use \*args and \*\*kwargs — But Parsimoniously

In essence, we use \*args to capture all (or an undetermined number of, to be more general) position arguments while \*\*kwargs to capture all (or an undetermined number of, to be more general) keyword arguments. Specifically, position arguments are based on the positions of the arguments that are passed in the function call, while keyword arguments are based on setting parameters to specifically named function arguments.

*>>> # Define a function that accepts undetermined position arguments*

*>>> def stringify(\*args):*

*... return [str(x) for x in args]*

*...*

*>>> stringify(2, False, None)*

*['2', 'False', 'None']*

The following code shows you a legitimate use of \*\*kwargs in function declaration.

*>>> # Define a function that accepts undetermined keyword arguments*

*... def generate\_score\_reports(name, \*\*kwargs):*

*... print(f"\*\*\*\*\* Report for {name} \*\*\*\*\*")*

*... for key, value in kwargs.items():*

*... print(f"### {key}: {value}")*

*... print("\*\*\*\*\* Report End \*\*\*\*\*\n")*

*...*

*... scores = {"John": {"math": 99, "phys": 97},*

*... "Jan": {"math": 94, "bio": 98}}*

*...*

*>>> for name, scores in scores.items():*

*... generate\_score\_reports(name, \*\*scores)*

*...*

*\*\*\*\*\* Report for John \*\*\*\*\**

*### math: 99*

*### phys: 97*

*\*\*\*\*\* Report End \*\*\*\*\**

9. Type Annotation for Arguments

The major benefit of having type annotation is that some IDEs (e.g., PyCharm or Visual Studio Code) could use the annotations to check the type compatibility for you, so that when you or other users use your functions you can get proper hints.

**// Python VS Node.js (study)**

<https://radixweb.com/blog/nodejs-vs-python>

<https://www.freecodecamp.org/news/nodejs-vs-python-choosing-the-best-technology-to-develop-back-end-of-your-web-app/>

<https://www.simform.com/blog/nodejs-vs-python/>

<https://kinsta.com/blog/nodejs-vs-python/>

🡪Working of Python Interpreter vs Node.js Architecture

<https://www.netguru.com/blog/node-js-vs-python>

<https://www.techmagic.co/blog/node-js-vs-python-what-to-choose/>

<https://www.bacancytechnology.com/blog/nodejs-vs-python>

<https://medium.com/front-end-weekly/nodejs-vs-python-which-one-to-choose-for-2024-0477d3ab7d5a>

<https://positiwise.com/blog/node-js-vs-python>

<https://www.clickittech.com/developer/node-js-vs-python/>

**// Python – Basics (syllabus)**

<https://www.aimoretechnologies.com/python-course-syllabus/>

<https://www.sevenmentor.com/wp-content/uploads/2019/04/Python-syllabus-2019-PDF-Download-.pdf>

<https://www.bestjavaclassesinpune.com/wp-content/uploads/2017/12/python-1.pdf>

<https://www.besanttechnologies.com/python-course-syllabus>

<https://www.scribd.com/document/451256132/Python-Core-and-Advanced-Syllabus>

<https://roadmap.sh/python>

**🡪Method Chaining**

from typing import Self

class MethodChainingExample:

def \_\_init\_\_(self) -> None: self.value = 0

def add(self, num: int) -> Self:

self.value += num; return self

def multiply(self, num: int) -> Self:

self.value \*= num; return self

def subtract(self, num: int) -> Self:

self.value -= num; return self

# Creating an instance of MethodChainingExample

method\_chaining\_example = MethodChainingExample()

# Method chaining example

result = method\_chaining\_example.add(5).multiply(3).subtract(2)

print(result.value) # Output: 13

**🡪asyncio py : async/await**

**https://docs.python.org/3/library/asyncio.html**

Started from 3.7 matured 3.11

The executor is given a timeout duration of 5 minutes to shutdown. If the executor hasn’t finished within that duration, a warning is emitted and the executor is closed.

**Coroutines**

a coroutine function: an async def function;

a coroutine object: an object returned by calling a coroutine function.

Tasks are used to schedule coroutines concurrently.

When a coroutine is wrapped into a Task with functions like asyncio.create\_task() the coroutine is automatically scheduled to run soon

1. **run()**

async def main(): await …  
asyncio.run(main())

1. **create\_task()**

**impot asyncio; import time**

**async** def main():

task1 = **asyncio.create\_task**(say\_after(1, 'hello'))

task2 = asyncio.create\_task(say\_after(2, 'world'))

print(f"started at {time.strftime('%X')}")

# Wait until both tasks are completed

**await task1**

await task2

print(f"finished at {time.strftime('%X')}")

1. **asyncioTaskGroup started in 3.11 py**

*https://docs.python.org/3/library/asyncio-task.html#asyncio.TaskGroup*

async def main(): # no need to use await in this pattern

**async with asyncio.TaskGroup() as tg**:

task1 = **tg.create\_task**(some\_coro(...))

task2 = tg.create\_task(another\_coro(...))

print(f"Both tasks have completed now: {task1.result()}, {task2.result()}")

1. **Awaitables**

print(await nested()) # will print "42".

1. **Futures**

A Future is a special low-level awaitable object that represents an eventual result of an asynchronous operation.

When a Future object is awaited it means that the coroutine will wait until the Future is resolved in some other place.

# this is also valid:

await asyncio.gather(

function\_that\_returns\_a\_future\_object(),

some\_python\_coroutine() )

**// Python – Intermediate (syllabus)**

<https://www.isoeh.com/Advanced%20Python%20Programming%20Syllabus.pdf>

<https://roadmap.sh/backend>

**// Python – Libraries (syllabus)**

🡪 Alternate python packages

<https://docs.python.org/3.12/using/windows.html#alternative-bundles>

Besides the standard CPython distribution, there are modified packages including additional functionality. The following is a list of popular versions and their key features:

ActivePython

Installer with multi-platform compatibility, documentation, PyWin32

Anaconda

Popular scientific modules (such as numpy, scipy and pandas) and the conda package manager.

Enthought Deployment Manager

“The Next Generation Python Environment and Package Manager”.

Previously Enthought provided Canopy, but it reached end of life in 2016.

WinPython

Windows-specific distribution with prebuilt scientific packages and tools for building packages.

Note that these packages may not include the latest versions of Python or other libraries, and are not maintained or supported by the core Python team.

🡪Top 10 Python Libraries

<https://www.interviewbit.com/blog/python-libraries/>

<https://tryolabs.com/blog/top-python-libraries-2023>

<https://www.geeksforgeeks.org/python-libraries-to-know/>

<https://geekflare.com/popular-python-libraries-modules/>

<https://www.quora.com/What-are-your-favorite-python-libraries>

<https://icore.sg/top-python-web-development-libraries/>

🡪Top 30 Python Libraries To Know

<https://www.mygreatlearning.com/blog/open-source-python-libraries/>

<https://hackr.io/blog/best-python-libraries>

<https://www.datacamp.com/blog/top-python-libraries-for-data-science>

<https://www.stxnext.com/blog/most-popular-python-scientific-libraries/>

🡪Top 40

<https://dev.to/taipy/top-42-python-libraries-you-need-to-know-1omo>

🡪Top 90+ Python Libraries

<https://flexiple.com/python/python-libraries>

**// Python – Frameworks (syllabus)**

<https://github.com/ml-tooling/best-of-web-python>

<https://wiki.python.org/moin/WebFrameworks>

[www.intel.com/content/www/us/en/developer/tools/oneapi/distribution-for-python.html](http://www.intel.com/content/www/us/en/developer/tools/oneapi/distribution-for-python.html)

<https://www.projectpro.io/article/python-libraries-for-web-scraping/625>

<https://research.aimultiple.com/python-web-scraping-libraries/>

<https://www.nobledesktop.com/outlines/python-web-development-with-django-syllabus.pdf>

<https://www.sevenmentor.com/wp-content/uploads/2023/04/SevenMentor-Django-Syllabus.pdf>

<https://datatechs.in/courses/django/>

<https://www.progclasses.com/Documentation/django-syllabus.pdf>

<https://clouddevs.com/python/libraries-for-web-development/>

<https://insights.daffodilsw.com/blog/top-10-python-frameworks-for-web-application-development>

<https://www.netsolutions.com/insights/top-10-python-frameworks-for-web-development-in-2019/>

<https://www.browserstack.com/guide/top-python-web-development-frameworks>

<https://www.analyticsinsight.net/top-5-python-libraries-for-web-development/>

<https://www.nobledesktop.com/classes-near-me/blog/popular-python-frameworks-for-web-development>

<https://radixweb.com/blog/best-python-frameworks>

<https://www.linkedin.com/pulse/choose-right-path-best-python-frameworks-web-app-development-rq0cc>

<https://kinsta.com/blog/python-frameworks/>

<https://www.knowledgehut.com/blog/web-development/python-frameworks>

**// Python – Advance (syllabus)**

**// Python – Web Development (syllabus)**

**// Python – DS (syllabus)**

**Python Version important points**

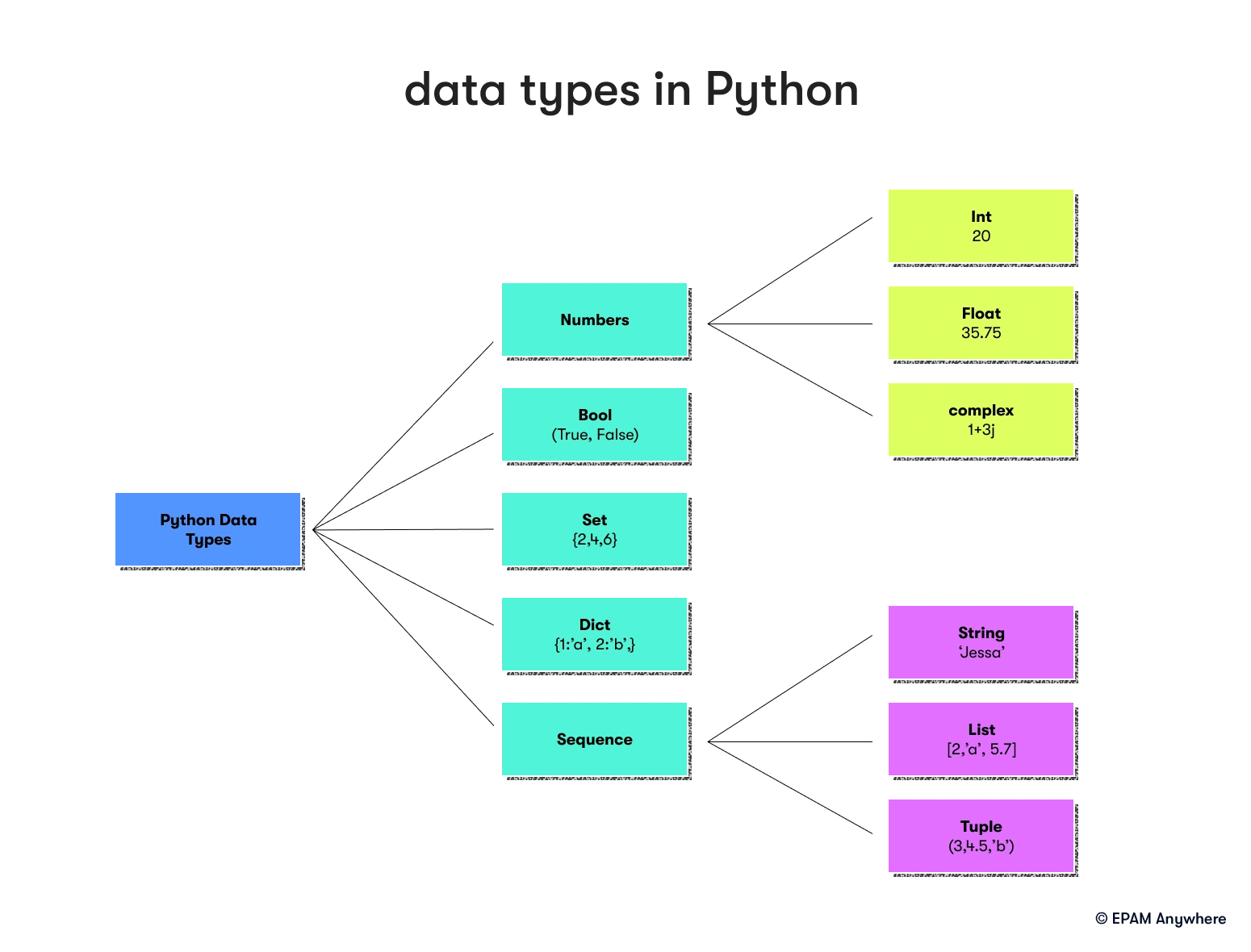
V3.12

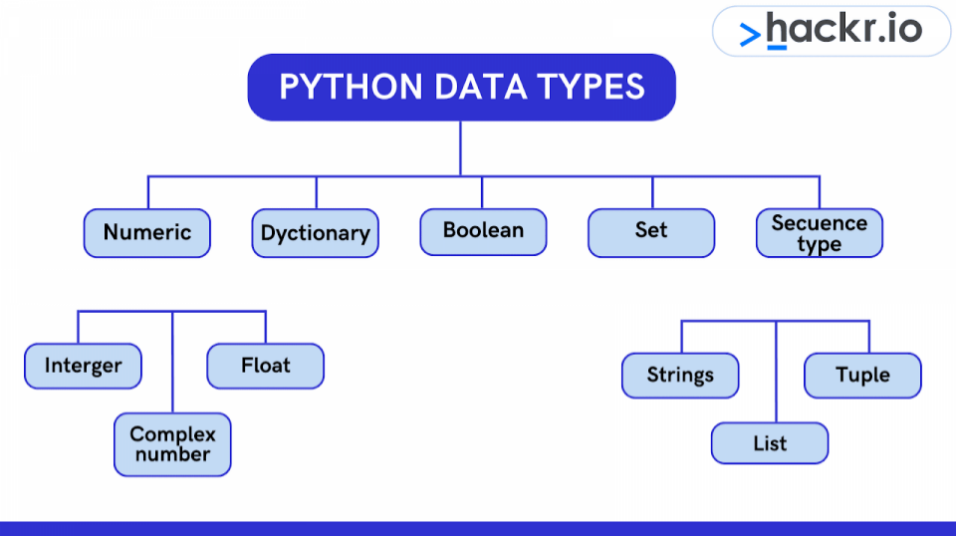
Removed:

* Do not pre-install setuptools in virtual environments created with venv. This means that distutils, setuptools, pkg\_resources, and easy\_install will no longer available by default; to access these run pip install setuptools in the activated virtual environment.

**Data model**

*https://docs.python.org/3/reference/datamodel.html*

From: <https://anywhere.epam.com/en/blog/python-interview-questions>



**Interview Question Set – Beginner**

<https://anywhere.epam.com/en/blog/python-interview-questions>

<https://www.mygreatlearning.com/blog/python-interview-questions/>

<https://hackr.io/blog/python-interview-questions>

<https://www.quora.com/What-are-popular-interview-questions-in-Python>

<https://medium.com/@nikitasilaparasetty/python-interview-questions-and-answers-for-beginners-f6aea485a686>

<https://www.bitdegree.org/tutorials/python-interview-questions>

<https://www.naukri.com/blog/python-interview-questions-for-freshers/>

<https://www.datacamp.com/blog/top-python-interview-questions-and-answers>

<https://medium.com/@nikitasilaparasetty/python-interview-coding-questions-with-solutions-for-beginners-7f6d782defac>

**Interview Question Set – Intermediate**

<https://www.mygreatlearning.com/blog/python-interview-questions/#python-interview-questions-for-experienced>

<https://iies.in/wp-content/uploads/2021/07/Interview-level-QA-on-Python-Programming.pdf>

**Interview Question Set – Advance**

**Interview Question Set – All Level**

<https://www.javatpoint.com/python-coding-interview-questions>

<https://www.mygreatlearning.com/blog/python-interview-questions/#python-interview-related-faqs>

<https://www.shiksha.com/online-courses/articles/python-interview-questions-answers/>

<https://www.scholarhat.com/tutorial/python/python-interview-questions-and-answers>

<https://hackr.io/blog/python-interview-questions>

<https://www.linkedin.com/pulse/top-100-python-interview-questions-answers-1f>

<https://blog.imocha.io/python-interview-questions-and-answers>

<https://brainstation.io/career-guides/python-developer-interview-questions>

<https://www.educative.io/blog/python-interview-questions>

<https://www.geeksforgeeks.org/python-interview-questions/>

<https://www.edureka.co/blog/interview-questions/python-interview-questions/>

<https://www.interviewbit.com/python-interview-questions/>

<https://intellipaat.com/blog/interview-question/python-interview-questions/>

<https://www.koenig-solutions.com/blog/top-python-interview-questions-and-answers>

<https://mindmajix.com/python-interview-questions>

https://github.com/Tanu-N-Prabhu/Python/blob/master/Python%20Coding%20Interview%20Prep/Python%20Coding%20Interview%20Questions%20(Beginner%20to%20Advanced).md

https://github.com/Tanu-N-Prabhu/Python

**Code practice questions**

<https://programiz.pro/course/python-interview-questions>

**Python code practice tools**

**Python online study portals**

<https://runestone.academy/ns/books/published/py4e-int/index.html>

<https://packaging.python.org/en/latest/>

<https://wiki.python.org/moin/PythonBooks>

<https://docs.python.org/3/>

<https://wiki.python.org/moin/BeginnersGuide>

<https://docs.python.org/3/library/>

<https://docs.python.org/3/tutorial/>

<https://docs.python.org/3/reference/>

<https://devguide.python.org/>

<https://www.python.org/about/gettingstarted/>

<https://www.tutorialspoint.com/python/index.htm>

<https://www.w3resource.com/python/python-tutorial.php>

<https://wiki.python.org/moin/WebProgramming>

<https://pythonprogramming.net/>

https://docs.python.org/2/howto/webservers.html