# Typing Constructs with Unexpected get\_origin() Results

Python’s typing.get\_origin() function usually returns the *unsubscripted* generic type for parameterized types. However, several standard typing constructs have edge-case behaviors where get\_origin() does not return the “expected” generic class or returns surprising values. Below are all such constructs, with examples of actual get\_origin() outputs:

* **typing.Annotated** (PEP 593): When using Annotated, get\_origin() returns the Annotated wrapper itself – **not** the underlying type. For example, get\_origin(Annotated[int, "meta"]) yields typing.Annotated[[1]](https://docs.python.org/uk/3/library/typing.html#:~:text=Note%20that%20using%20get_origin,itself). (By contrast, the \_\_origin\_\_ attribute of an Annotated type yields the wrapped base type, e.g. Annotated[int, ...].\_\_origin\_\_ is int[[2]](https://docs.python.org/uk/3/library/typing.html#:~:text=If%20you%20want%20to%20retrieve,attribute).) This means the origin is the Annotated class rather than the raw type.
* **typing.Optional and typing.Union** (PEP 604): Optional[T] is just an alias for Union[T, None]. Accordingly, get\_origin(Optional[int]) returns typing.Union. In general, get\_origin(Union[X, Y, ...]) returns the Union special form[[3]](https://docs.python.org/uk/3/library/typing.html#:~:text=assert%20get_origin,is%20Annotated). One quirk is that a *single* type union collapses to the type itself – e.g. Union[int] is just int, so get\_origin(Union[int]) returns None (no generic origin) in that degenerate case. Also note that using the new union syntax (X | Y) produces a types.UnionType internally (PEP 604). For instance, get\_origin(int | str) will return the internal types.UnionType class instead of typing.Union (since | creates a union at the core language level). This is an implementation detail – types.UnionType is essentially the runtime type for all union expressions[[4]](https://docs.python.org/3.10/library/typing.html#:~:text=Introducing%20Annotated)【8†】.
* **typing.ClassVar and typing.Final**: These are markers for special usage (class-level variables and final variables). For parameterized forms like ClassVar[T] or Final[T], get\_origin() returns the marker itself rather than None. For example, get\_origin(ClassVar[int]) yields typing.ClassVar[[5]](https://android.googlesource.com/platform/prebuilts/python/darwin-x86/+/refs/heads/busytown-mac1010-release/lib/python3.9/typing.py#:~:text=This%20supports%20generic%20types%2C%20Callable%2C,int%5D%29%20%3D%3D%20list)[[6]](https://android.googlesource.com/platform/prebuilts/python/darwin-x86/+/refs/heads/busytown-mac1010-release/lib/python3.9/typing.py#:~:text=get_origin%28int%29%20is%20None%20get_origin%28ClassVar,Generic%29%20is%20Generic), and get\_origin(Final[str]) similarly returns typing.Final. (Older implementations treated ClassVar as non-generic and would return None, but in Python 3.9+ this was changed[[7]](https://stackoverflow.com/questions/62074855/how-to-create-custom-types-using-typing-module#:~:text=Callable%2C%20and%20Tuple,if%20isinstance%28typ%2C%20_GenericAlias)[[8]](https://android.googlesource.com/platform/prebuilts/python/darwin-x86/+/refs/heads/busytown-mac1010-release/lib/python3.9/typing.py#:~:text=get_origin%28Literal,Generic%29%20is%20Generic).) The origin remains the special form (ClassVar/Final) instead of a normal class.
* **Literal Type Guards**: Constructs like typing.TypeGuard (PEP 647) follow the same pattern as Final/ClassVar. TypeGuard[T] is a single-parameter special form, and get\_origin(TypeGuard[int]) returns typing.TypeGuard itself (not the inner type). Similarly, the TypedDict field qualifiers Required[T] and NotRequired[T] (PEP 655) have origins equal to themselves (e.g. get\_origin(Required[int]) is typing.Required) since they are treated as special typing forms.
* **Normalization to Built-in Classes**: Many typing aliases for collections or other generics resolve to actual concrete classes. Per design, if the origin is a standard library class, get\_origin() yields that real class[[9]](https://docs.python.org/uk/3/library/typing.html#:~:text=If%20%60X%60%20is%20a%20typing,for%20unsupported%20objects)[[10]](https://docs.python.org/uk/3/library/typing.html#:~:text=assert%20get_origin%28Dict,P). For example, get\_origin(List[str]) is <class 'list'> and get\_origin(Dict[str, int]) is <class 'dict'>[[11]](https://runebook.dev/ja/docs/python/library/typing#:~:text=assert%20get_origin,is%20Annotated). The same applies to other built-in generics like Tuple (origin becomes tuple), Set (set), FrozenSet (frozenset), etc. In each case the typing alias is “normalized” to its corresponding built-in type[[9]](https://docs.python.org/uk/3/library/typing.html#:~:text=If%20%60X%60%20is%20a%20typing,for%20unsupported%20objects)[[11]](https://runebook.dev/ja/docs/python/library/typing#:~:text=assert%20get_origin,is%20Annotated). Likewise, typing.Callable and typing.Type are normalized: for instance, get\_origin(Callable[[int, str], bool]) yields collections.abc.Callable (the abstract base class) and get\_origin(Type[MyClass]) returns the built-in type object【10†】【11†】. In these cases, the origin is a concrete class or ABC, not a typing pseudo-type – which may be unexpected if one anticipated the typing alias itself.
* **ParamSpec “pseudo-parameters”** (PEP 612): If you use a ParamSpec in variadic form, the .args and .kwargs attributes act like special parameterized types. Calling get\_origin() on these returns the original ParamSpec object. For example, if P = ParamSpec('P'), then get\_origin(P.args) is P (the ParamSpec)[[12]](https://docs.python.org/uk/3/library/typing.html#:~:text=If%20%60X%60%20is%20a%20typing,for%20unsupported%20objects)[[13]](https://docs.python.org/uk/3/library/typing.html#:~:text=assert%20get_origin%28Union,args%29%20is%20P). This is a unique case where the “origin” is neither a class nor a typing form, but the ParamSpec type variable itself.
* **typing.NewType**: NewType creates a callable that’s essentially an identity function for a base type. A NewType is not a generic alias at all – it doesn’t support subscripting – so get\_origin() will return None for a NewType object. For example, UserId = NewType('UserId', int) produces a custom type function, and get\_origin(UserId) is None (there is no generic origin). Even though NewType wraps an underlying type, it isn’t a parameterized typing construct, so it’s treated as “unsupported” by get\_origin[[7]](https://stackoverflow.com/questions/62074855/how-to-create-custom-types-using-typing-module#:~:text=Callable%2C%20and%20Tuple,if%20isinstance%28typ%2C%20_GenericAlias)【15†】.

**Sources:** The behaviors above are confirmed by Python’s documentation and examples. The typing module docs note that get\_origin() returns the outermost type for special forms (e.g. Annotated, ClassVar, Final, Literal) and normalizes aliases to built-ins[[14]](https://runebook.dev/ja/docs/python/library/typing#:~:text=assert%20get_origin,is%20Annotated)[[1]](https://docs.python.org/uk/3/library/typing.html#:~:text=Note%20that%20using%20get_origin,itself). They also illustrate that union types and Optional are handled via Union (or types.UnionType for the | syntax)[[15]](https://docs.python.org/uk/3/library/typing.html#:~:text=%D0%BF%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%B8%3A)【8†】. Cases like ParamSpec arguments returning the ParamSpec, and unsupported cases returning None, are likewise documented or observed in runtime behavior[[12]](https://docs.python.org/uk/3/library/typing.html#:~:text=If%20%60X%60%20is%20a%20typing,for%20unsupported%20objects)[[5]](https://android.googlesource.com/platform/prebuilts/python/darwin-x86/+/refs/heads/busytown-mac1010-release/lib/python3.9/typing.py#:~:text=This%20supports%20generic%20types%2C%20Callable%2C,int%5D%29%20%3D%3D%20list). Each example above shows an instance where get\_origin()’s result deviates from a naive expectation of the “generic” type, making these constructs important edge cases.

[[1]](https://docs.python.org/uk/3/library/typing.html#:~:text=Note%20that%20using%20get_origin,itself) [[2]](https://docs.python.org/uk/3/library/typing.html#:~:text=If%20you%20want%20to%20retrieve,attribute) [[3]](https://docs.python.org/uk/3/library/typing.html#:~:text=assert%20get_origin,is%20Annotated) [[9]](https://docs.python.org/uk/3/library/typing.html#:~:text=If%20%60X%60%20is%20a%20typing,for%20unsupported%20objects) [[10]](https://docs.python.org/uk/3/library/typing.html#:~:text=assert%20get_origin%28Dict,P) [[12]](https://docs.python.org/uk/3/library/typing.html#:~:text=If%20%60X%60%20is%20a%20typing,for%20unsupported%20objects) [[13]](https://docs.python.org/uk/3/library/typing.html#:~:text=assert%20get_origin%28Union,args%29%20is%20P) [[15]](https://docs.python.org/uk/3/library/typing.html#:~:text=%D0%BF%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%B8%3A) typing — Support for type hints — Python 3.13.7 documentation

<https://docs.python.org/uk/3/library/typing.html>

[[4]](https://docs.python.org/3.10/library/typing.html#:~:text=Introducing%20Annotated) typing — Support for type hints — Python 3.10.18 documentation

<https://docs.python.org/3.10/library/typing.html>

[[5]](https://android.googlesource.com/platform/prebuilts/python/darwin-x86/+/refs/heads/busytown-mac1010-release/lib/python3.9/typing.py#:~:text=This%20supports%20generic%20types%2C%20Callable%2C,int%5D%29%20%3D%3D%20list) [[6]](https://android.googlesource.com/platform/prebuilts/python/darwin-x86/+/refs/heads/busytown-mac1010-release/lib/python3.9/typing.py#:~:text=get_origin%28int%29%20is%20None%20get_origin%28ClassVar,Generic%29%20is%20Generic) [[8]](https://android.googlesource.com/platform/prebuilts/python/darwin-x86/+/refs/heads/busytown-mac1010-release/lib/python3.9/typing.py#:~:text=get_origin%28Literal,Generic%29%20is%20Generic) lib/python3.9/typing.py - platform/prebuilts/python/darwin-x86 - Git at Google

<https://android.googlesource.com/platform/prebuilts/python/darwin-x86/+/refs/heads/busytown-mac1010-release/lib/python3.9/typing.py>

[[7]](https://stackoverflow.com/questions/62074855/how-to-create-custom-types-using-typing-module#:~:text=Callable%2C%20and%20Tuple,if%20isinstance%28typ%2C%20_GenericAlias) python - How to create custom types using typing module? - Stack Overflow

<https://stackoverflow.com/questions/62074855/how-to-create-custom-types-using-typing-module>

[[11]](https://runebook.dev/ja/docs/python/library/typing#:~:text=assert%20get_origin,is%20Annotated) [[14]](https://runebook.dev/ja/docs/python/library/typing#:~:text=assert%20get_origin,is%20Annotated) Python typing 日本語

<https://runebook.dev/ja/docs/python/library/typing>