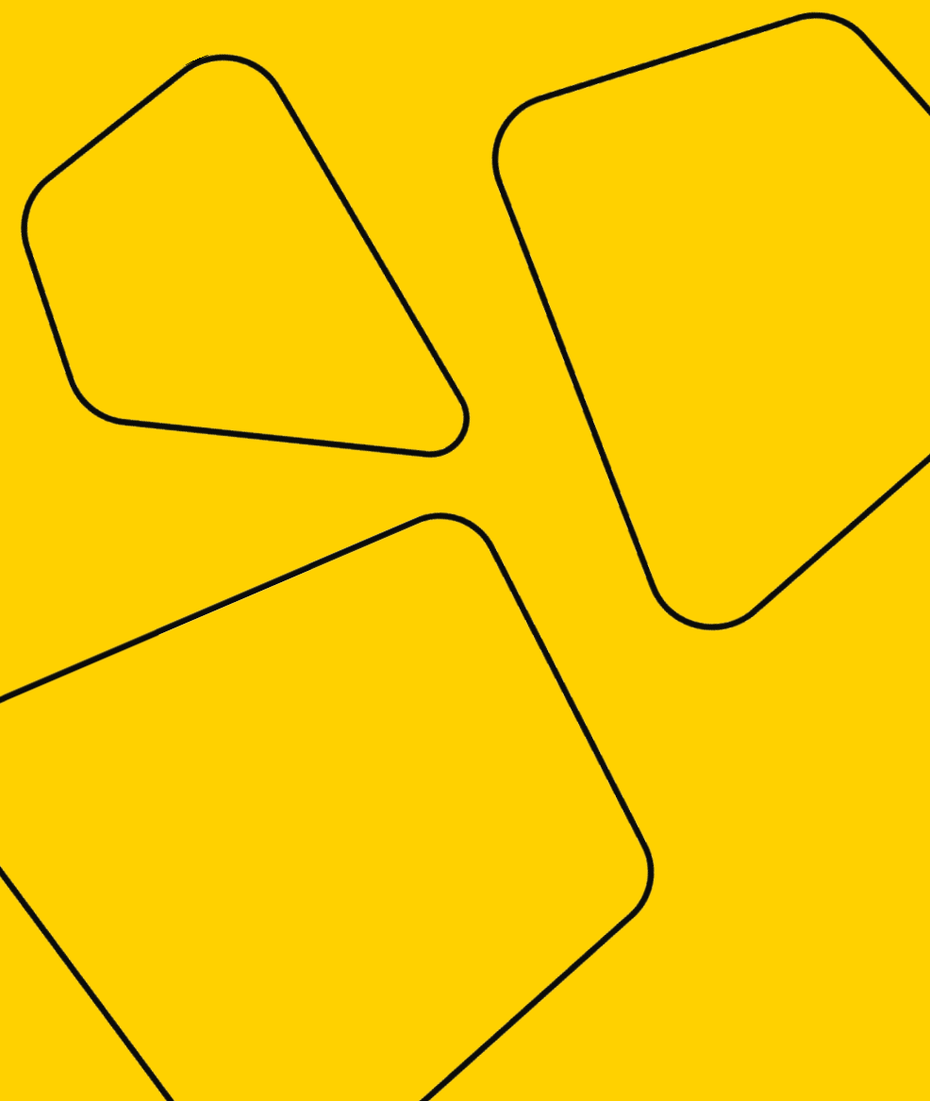
Classes

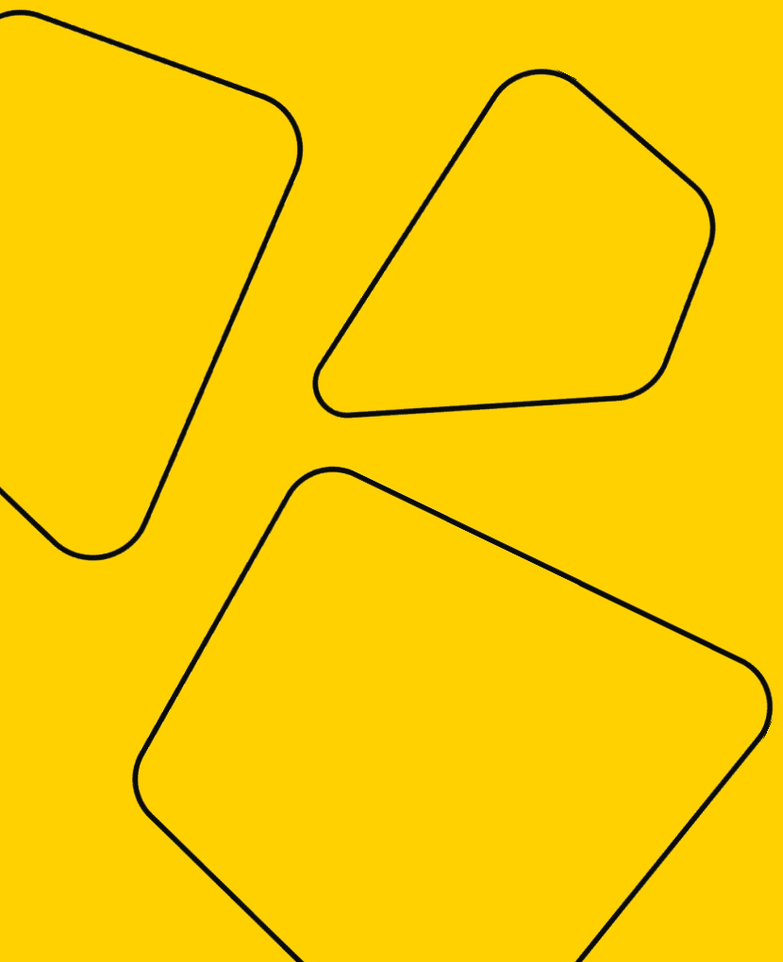
Software Development & Python

Nick Levashov, 2021



girafe
ai

Classes



```
@add_print_method
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Git
GitHub

```
$ git init
$ git add
$ git commit
$ git push

$ git status
$ git commit
$ git push
```

GitHub

PyCharm

The Python IDE
for Professional Developers

Class definition

girafe
ai

Empty class

```
class ClassName: pass
```

Empty class

```
class ClassName: pass
```

```
CamelCase

snake_case
```

Class methods

```
class PingPong:

    def ping(self):
        return 'pong'
```

Class methods

```
class PingPong:

    def ping(self):
        return 'pong'
```

```
>>> pp = PingPong()
>>> pp.ping()
'pong'
```

“self” and instance attributes

```
class PingPong:

    def ping(self):
        self.last_result = 'pong'
        return 'pong'

    def pong(self):
        self.last_result = 'ping'
        return 'ping'
```

“self” and instance attributes

```
class PingPong:

    def ping(self):
        self.last_result = 'pong'
        return 'pong'

    def pong(self):
        self.last_result = 'ping'
        return 'ping'
```

```
>>> pp1 = PingPong()
>>> pp1.ping()
'pong'
>>> pp2 = PingPong()
>>> pp2.pong()
'ping'
```

“self” and instance attributes

```
class PingPong:

    def ping(self):
        self.last_result = 'pong'
        return 'pong'

    def pong(self):
        self.last_result = 'ping'
        return 'ping'


>>> pp1 = PingPong()
>>> pp1.ping()
'pong'
>>> pp2 = PingPong()
>>> pp2.pong()
'ping'
>>> print(pp1.last_result, pp2.last_result)
pong ping
```

Class initialization

```
class StrList:
    def __init__(self):
        self.state = []
```

Class initialization

```
class StrList:
    def __init__(self, state: list[str] | None = None):
        self.state = state or []
```

Class initialization

```
class StrList:

    def __init__(self, state: list[str] | None = None):
        if not state:
            self.state = []
            return

        if not isinstance(state, list):
            raise TypeError('state must be list[str]')

        for element in state:
            if not isinstance(element, str):
                raise TypeError('state must be list[str]')
```

Class initialization

```
class StrList:

    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)

    def validate_state(self, state: list[str] | None) -> list[str]:
        if not state:
            return []

        if not isinstance(state, list):
            raise TypeError('state must be list[str]')

        for element in state:
            if not isinstance(element, str):
                raise TypeError('state must be list[str]')

        return state
```

Class initialization

```
class StrList:
    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)
    def validate_state(self, state: list[str] | None) -> list[str]:
        if not state:
            return []
        if not isinstance(state, list):
            raise TypeError('state must be list[str]')
        for element in state:
            if not isinstance(element, str):
                raise TypeError('state must be list[str]')
        return state
```

```
>>> sl = StrList()
>>> sl.state
[]
```

Class initialization

```
class StrList:
    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)
    def validate_state(self, state: list[str] | None) -> list[str]:
        if not state:
            return []
        if not isinstance(state, list):
            raise TypeError('state must be list[str]')
        for element in state:
            if not isinstance(element, str):
                raise TypeError('state must be list[str]')
        return state


>>> StrList().state
[]
```

Class initialization

```
class StrList:
    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)
    def validate_state(self, state: list[str] | None) -> list[str]:
        if not state:
            return []
        if not isinstance(state, list):
            raise TypeError('state must be list[str]')
        for element in state:
            if not isinstance(element, str):
                raise TypeError('state must be list[str]')
        return state
```

```
>>> StrList().state
[]
>>> StrList([]).state
[]
```

Class initialization

```
class StrList:
    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)
    def validate_state(self, state: list[str] | None) -> list[str]:
        if not state:
            return []
        if not isinstance(state, list):
            raise TypeError('state must be list[str]')
        for element in state:
            if not isinstance(element, str):
                raise TypeError('state must be list[str]')
        return state
```

```
>>> StrList().state
[]
>>> StrList([]).state
[]
>>> StrList(['1']).state
['1']
```

Class initialization

```
class StrList:
    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)
    def validate_state(self, state: list[str] | None) -> list[str]:
        if not state:
            return []
        if not isinstance(state, list):
            raise TypeError('state must be list[str]')
        for element in state:
            if not isinstance(element, str):
                raise TypeError('state must be list[str]')
        return state
```

```
>>> StrList().state
[]
>>> StrList([]).state
[]
>>> StrList(['1']).state
['1']
>>> StrList(['abc', 'def', 'ghi']).state
['abc', 'def', 'ghi']
```

Class initialization

```
class StrList:
    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)
    def validate_state(self, state: list[str] | None) -> list[str]:
        if not state:
            return []
        if not isinstance(state, list):
            raise TypeError('state must be list[str]')
        for element in state:
            if not isinstance(element, str):
                raise TypeError('state must be list[str]')
        return state
```

```
>>> StrList(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in __init__
  File "<stdin>", line 8, in validate_state
TypeError: state must be list[str]
```

Class initialization

```
class StrList:
    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)
    def validate_state(self, state: list[str] | None) -> list[str]:
        if not state:
            return []
        if not isinstance(state, list):
            raise TypeError('state must be list[str]')
        for element in state:
            if not isinstance(element, str):
                raise TypeError('state must be list[str]')
        return state
```

```
>>> StrList([1, 2, 3])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in __init__
  File "<stdin>", line 11, in validate_state
TypeError: state must be list[str]
```

Class initialization

```
class StrList:
    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)
    def validate_state(self, state: list[str] | None) -> list[str]:
        if not state:
            return []
        if not isinstance(state, list):
            raise TypeError('state must be list[str]')
        for element in state:
            if not isinstance(element, str):
                raise TypeError('state must be list[str]')
        return state
```

```
>>> StrList({'1'})
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in __init__
  File "<stdin>", line 8, in validate_state
TypeError: state must be list[str]
```

Class initialization

```
class StrList:

    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)

    def validate_state(self, state: list[str] | None) -> list[str]:
        if state is None:
            return []

        if not isinstance(state, list):
            raise TypeError('state must be list[str]')

        for element in state:
            if not isinstance(element, str):
                raise TypeError('state must be list[str]')

        return state
```

Class constructor

```python
class StrList:

    def __new__(cls, state: list[str] | None = None):
        return  # TODO create class instance

    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)

    def validate_state(self, state: list[str] | None) -> list[str]: ...
```

Class methods

```
class StrList:

    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)

    def validate_state(self, state: list[str] | None) -> list[str]: ...

    def append(self, element: str) -> None:
        if not isinstance(element, str):
            raise TypeError('element must be str')
    self.state.append(element)
```

Class methods

```
class StrList:

    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)

    def validate_state(self, state: list[str] | None) -> list[str]: ...

    def validate_str(self, element: str) -> str:
        if not isinstance(element, str):
            raise TypeError('element must be str')
        return element

def append(self, element: str) -> None:
    self.state.append(self.validate_str(element))
```

Class methods

```python
class StrList:

    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)

    def validate_state(self, state: list[str] | None) -> list[str]: ...

    def validate_str(self, element: str) -> str:
        if not isinstance(element, str):
            raise TypeError('element must be str')
        return element

def append(self, element: str) -> None:
    self.state.append(self.validate_str(element))

def extend(self, state: list[str] | None = None) -> None:
    self.state += self.validate_state(state)
```

Class methods

```python
class StrList:

    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)

    def validate_state(self, state: list[str] | None) -> list[str]: ...

    def validate_str(self, element: str) -> str:
        if not isinstance(element, str):
            raise TypeError('element must be str')
        return element

    def append(self, element: str) -> None:
        self.state.append(self.validate_str(element))

    def extend(self, state: list[str] | None = None) -> None:
        self.state += self.validate_state(state)

    def remove(self, element: str) -> None:
        self.state.remove(element)
```

Class methods

```
class StrList:

    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)

    def validate_state(self, state: list[str] | None) -> list[str]: ...
    def validate_str(self, element: str) -> str: ...

    def append(self, element: str) -> None:
        self.state.append(self.validate_str(element))
    def extend(self, state: list[str] | None = None) -> None:
        self.state += self.validate_state(state)
    def remove(self, element: str) -> None:
        self.state.remove(element)
```

```
>>> sl = StrList(['a', 'b'])
>>> sl.append('c')
>>> sl.extend(['d', 'e'])
>>> sl.remove('c')
>>> sl.state
['a', 'b', 'd', 'e']
```

Class attributes

```
class StrList:

    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)

    def validate_state(self, state: list[str] | None) -> list[str]: ...
    def validate_str(self, element: str) -> str: ...
    def append(self, element: str) -> None: ...
    def extend(self, state: list[str] | None = None) -> None: ...
    def remove(self, element: str) -> None: ...
```

```
    def print(self):
        n = 3
        elements_to_print = self.state[:n]
        print(f'StrList({len(self.state)}): {elements_to_print}')
```

Class attributes

```
class StrList:
    PRINT_COUNT = 3

    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)

    def validate_state(self, state: list[str] | None) -> list[str]: ...
    def validate_str(self, element: str) -> str: ...
    def append(self, element: str) -> None: ...
    def extend(self, state: list[str] | None = None) -> None: ...
    def remove(self, element: str) -> None: ...

def print(self):
    elements_to_print = self.state[:self.PRINT_COUNT]
    print(f'StrList({len(self.state)}): {elements_to_print}')
```

Class attributes

```
class StrList:
    PRINT_COUNT = 3

    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)

    def validate_state(self, state: list[str] | None) -> list[str]: ...
    def validate_str(self, element: str) -> str: ...
    def append(self, element: str) -> None: ...
    def extend(self, state: list[str] | None = None) -> None: ...
    def remove(self, element: str) -> None: ...

    def print(self):
        elements_to_print = self.state[:self.PRINT_COUNT]
        print(f'StrList({len(self.state)}): {elements_to_print}')
```

```
>>> sl = StrList(['a', 'b', 'c', 'd', 'e'])
>>> sl.print()
StrList(5): ['a', 'b', 'c']
```

Class attributes

```
class StrList:
    PRINT_COUNT = 3

    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)

    def validate_state(self, state: list[str] | None) -> list[str]: ...
    def validate_str(self, element: str) -> str: ...
    def append(self, element: str) -> None: ...
    def extend(self, state: list[str] | None = None) -> None: ...
    def remove(self, element: str) -> None: ...

    def print(self):
        elements_to_print = self.state[:self.PRINT_COUNT]
        print(f'StrList({len(self.state)}): {elements_to_print}')
```

```
>>> StrList.__dict__
mappingproxy({'__module__': '__main__', 'PRINT_COUNT': 3, '__init__':
<function StrList.__init__ at 0x1100d7910>, 'print': <function
StrList.print at 0x1100d79a0>, '__dict__': <attribute '__dict__' of
'StrList' objects>, '__weakref__': <attribute '__weakref__' of 'StrList'
objects>, '__doc__': None})

>>> StrList(['a', 'b', 'c']).__dict__
{'state': ['a', 'b', 'c']}
```

classmethod and staticmethod

```
class StrList:
    ...

    @classmethod
    def validate_state(cls, state: list[str] | None) -> list[str]:
        if state is None:
            return []

        if not isinstance(state, list):
            raise TypeError(f'state of {cls.__name__} must be list[str]')
        for element in state:
            if not isinstance(element, str):
                raise TypeError(f'state of {cls.__name__} must be list[str]')

        return state

    @staticmethod
    def validate_str(element: str) -> str:
        if not isinstance(element, str):
            raise TypeError('element must be str')
        return element
```

classmethod and staticmethod

```
class StrList:
    @classmethod
     def validate_state(cls, state: list[str] | None) -> list[str]:
         if state is None:
             return []
         if not isinstance(state, list):
             raise TypeError(f'state of {cls.__name__} must be list[str]')
         for element in state:
             if not isinstance(element, str):
                 raise TypeError(f'state of {cls.__name__} must be list[str]')
         return state

     @staticmethod
     def validate_str(element: str) -> str:
         if not isinstance(element, str):
             raise TypeError('element must be str')
         return element

>>> sl = StrList(['1', '2', 3])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in __init__
  File "<stdin>", line 14, in validate_state
TypeError: state of StrList must be list[str]
```

classmethod and staticmethod

```
class StrList:
    @classmethod
     def validate_state(cls, state: list[str] | None) -> list[str]:
         if state is None:
             return []
         if not isinstance(state, list):
             raise TypeError(f'state of {cls.__name__} must be list[str]')
         for element in state:
             if not isinstance(element, str):
                 raise TypeError(f'state of {cls.__name__} must be list[str]')
         return state

     @staticmethod
     def validate_str(element: str) -> str:
         if not isinstance(element, str):
             raise TypeError('element must be str')
         return element

>>> sl = StrList(['1', '2', '3'])
>>> sl.append(4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 5, in append
  File "<stdin>", line 19, in validate_str
TypeError: element must be str
```

classmethod and staticmethod

```
class StrList:
    @classmethod
     def validate_state(cls, state: list[str] | None) -> list[str]:
         if state is None:
             return []
         if not isinstance(state, list):
             raise TypeError(f'state of {cls.__name__} must be list[str]')
         for element in state:
             if not isinstance(element, str):
                 raise TypeError(f'state of {cls.__name__} must be list[str]')
         return state

    @staticmethod
    def validate_str(element: str) -> str:
         if not isinstance(element, str):
             raise TypeError('element must be str')
         return element


>>> sl.validate_state(['1', '2', '3'])
['1', '2', '3']
>>> sl.validate_str('4')
'4'
```

classmethod and staticmethod

```
class StrList:
    @classmethod
     def validate_state(cls, state: list[str] | None) -> list[str]:
         if state is None:
             return []
         if not isinstance(state, list):
             raise TypeError(f'state of {cls.__name__} must be list[str]')
         for element in state:
             if not isinstance(element, str):
                 raise TypeError(f'state of {cls.__name__} must be list[str]')
         return state

     @staticmethod
     def validate_str(element: str) -> str:
         if not isinstance(element, str):
             raise TypeError('element must be str')
         return element


>>> StrList.validate_state(['1', '2', '3'])
['1', '2', '3']
>>> StrList.validate_str('4')
'4'
```

Internal variables

	Different class but same package	Different package but subclass	Unrelated class but same module	Different module and p1 not exported
`package p1;`	`package p1;`	`package p2;`	`package p2;`	`package x;`
`class A {`	`class B {`	`class C extends A`	`class D {`	`class E {`
`private int i;`		`{`		
`int j;`				
`protected int k;`				
`public int l;`				
`}`	`}`		`}`	`}`

Accessible Inaccessible

Internal variables

```
class StrList:
    def __init__(self, state: list[str] | None = None):
        self.state = self.validate_state(state)

    def validate_state(cls, state: list[str] | None) -> list[str]: ...
    def validate_str(element: str) -> str: ...

    def append(self, element: str) -> None: ...
    def extend(self, state: list[str] | None = None) -> None: ...
    def remove(self, element: str) -> None: ...
    def print(self): ...
```

Internal variables

```python
class StrList:
    def __init__(self, state: list[str] | None = None):
        self._state = self._validate_state(state)

    def _validate_state(cls, state: list[str] | None) -> list[str]: ...
    def _validate_str(element: str) -> str: ...

    def append(self, element: str) -> None: ...
    def extend(self, state: list[str] | None = None) -> None: ...
    def remove(self, element: str) -> None: ...
    def print(self): ...
```

Internal variables

```
class StrList:
    def __init__(self, state: list[str] | None = None):
        self.__state = self.__validate_state(state)

    def __validate_state(cls, state: list[str] | None) -> list[str]: ...
    def __validate_str(element: str) -> str: ...

    def append(self, element: str) -> None: ...
    def extend(self, state: list[str] | None = None) -> None: ...
    def remove(self, element: str) -> None: ...
    def print(self): ...
```

Internal variables

```
class StrList:
    def __init__(self, state: list[str] | None = None):
        self.__state = self.__validate_state(state)

    def __validate_state(cls, state: list[str] | None) -> list[str]: ...
    def __validate_str(element: str) -> str: ...

    def append(self, element: str) -> None: ...
    def extend(self, state: list[str] | None = None) -> None: ...
    def remove(self, element: str) -> None: ...
    def print(self): ...
```

```
>>> dir(StrList())
['_StrList__state', '_StrList__validate_state', '_StrList__validate_str',
'__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
'__init__', '__init_subclass__', '__le__', '__lt__', '__module__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'__weakref__', 'append', 'extend', 'print', 'remove']
```

Internal variables

```
class StrList:
    def __init__(self, state: list[str] | None = None):
        self.__state = self.__validate_state(state)

    def __validate_state(cls, state: list[str] | None) -> list[str]: ...
    def __validate_str(element: str) -> str: ...

    def append(self, element: str) -> None: ...
    def extend(self, state: list[str] | None = None) -> None: ...
    def remove(self, element: str) -> None: ...
    def print(self): ...


>>> StrList().__state
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'StrList' object has no attribute '__state'
```

Internal variables

```
class StrList:
    def __init__(self, state: list[str] | None = None):
        self.__state = self.__validate_state(state)

    def __validate_state(cls, state: list[str] | None) -> list[str]: ...
    def __validate_str(element: str) -> str: ...

    def append(self, element: str) -> None: ...
    def extend(self, state: list[str] | None = None) -> None: ...
    def remove(self, element: str) -> None: ...
    def print(self): ...


>>> StrList()._StrList__state
[]
```

Internal variables

```
class StrList:
    def __init__(self, state: list[str] | None = None):
        self._state = self._validate_state(state)

    def _validate_state(cls, state: list[str] | None) -> list[str]: ...
    def _validate_str(element: str) -> str: ...

    def get_state(self) -> list[str]:
        return self._state
```

```
def append(self, element: str) -> None: ...
def extend(self, state: list[str] | None = None) -> None: ...
def remove(self, element: str) -> None: ...
def print(self): ...
```

Property

```
class StrList:
    def __init__(
            self,
            state: list[str] | None = None,
            max_length: int | None = None,
    ):
        self._state = self._validate_state(state)
        self._max_length = \
            self._validate_int(max_length) if max_length is not None else None
        self._state = self._state[:self._max_length]
```

...

Property

```
class StrList:
    def __init__(
            self,
            state: list[str] | None = None,
            max_length: int | None = None,
    ):
        self._state = self._validate_state(state)
        self._max_length = \
            self._validate_int(max_length) if max_length is not None else None
        self._state = self._state[:self._max_length]

    @property
    def max_length(self):
        return self._max_length

    ...


>>> sl = StrList(max_length=10)
>>> sl.max_length
10
```

Property: getter, setter, deleter

```
class StrList:
    def __init__(
            self,
            state: list[str] | None = None,
            max_length: int | None = None,
    ):
        self._state = self._validate_state(state)
        self._max_length = \
            self._validate_int(max_length) if max_length is not None else None
        self._state = self._state[:self._max_length]
```

```
@property
def max_length(self):
    return self._max_length

@max_length.setter
def max_length(self, max_length):
    self._max_length = self._validate_int(max_length)
    self._state = self._state[:self._max_length]

@max_length.deleter
def max_length(self):
    self._max_length = None

...
```

Property: getter, setter, deleter

```
class StrList:
    ...
    @property
    def max_length(self):
        return self._max_length
    @max_length.setter
    def max_length(self, max_length):
        self._max_length = self._validate_int(max_length)
        self._state = self._state[:self._max_length]
    @max_length.deleter
    def max_length(self):
        self._max_length = None
    ...


>>> sl = StrList(max_length=10)
>>> print(sl.max_length)
10
>>> sl.max_length = 50
>>> print(sl.max_length)
50
>>> del sl.max_length
>>> print(sl.max_length)
None
```

Class definition

girafe
ai