Set Objects

This section details the public API for `set` and `frozenset` objects. Any functionality not listed below is best accessed using the either the abstract object protocol (including `PyObject_CallMethod()`, `PyObject_RichCompareBool()`, `PyObject_Hash()`, `PyObject_Repr()`, `PyObject_IsTrue()`, `PyObject_Print()`, and `PyObject_GetIter()`) or the abstract number protocol (including `PyNumber_And()`, `PyNumber_Subtract()`, `PyNumber_Or()`, `PyNumber_Xor()`, `PyNumber_InPlaceAnd()`, `PyNumber_InPlaceSubtract()`, `PyNumber_InPlaceOr()`, and `PyNumber_InPlaceXor()`).

`PySetObject`

This subtype of `PyObject` is used to hold the internal data for both `set` and `frozenset` objects. It is like a `PyDictObject` in that it is a fixed size for small sets (much like tuple storage) and will point to a separate, variable sized block of memory for medium and large sized sets (much like list storage). None of the fields of this structure should be considered public and are subject to change. All access should be done through the documented API rather than by manipulating the values in the structure.

PyTypeObject **`PySet_Type`**

This is an instance of `PyTypeObject` representing the Python `set` type.

PyTypeObject **`PyFrozenSet_Type`**

This is an instance of `PyTypeObject` representing the Python `frozenset` type.

The following type check macros work on pointers to any Python object. Likewise, the constructor functions work with any iterable Python object.

int **`PySet_Check`**(PyObject *p)

Return true if *p* is a `set` object or an instance of a subtype.

int **`PyFrozenSet_Check`**(PyObject *p)

Return true if *p* is a `frozenset` object or an instance of a subtype.

int **`PyAnySet_Check`**(PyObject *p)

Return true if *p* is a `set` object, a `frozenset` object, or an instance of a subtype.

int **`PyAnySet_CheckExact`**(PyObject *p)

Return true if *p* is a `set` object or a `frozenset` object but not an instance of a subtype.

int **`PyFrozenSet_CheckExact`**(PyObject *p)

Return true if *p* is a `frozenset` object but not an instance of a subtype.

PyObject* **PySet_New**(PyObject **iterable*)

Return value: New reference.

Return a new `set` containing objects returned by the *iterable*. The *iterable* may be *NULL* to create a new empty set. Return the new set on success or *NULL* on failure. Raise `TypeError` if *iterable* is not actually iterable. The constructor is also useful for copying a set (`c=set(s)`).

PyObject* **PyFrozenSet_New**(PyObject **iterable*)

Return value: New reference.

Return a new `frozenset` containing objects returned by the *iterable*. The *iterable* may be *NULL* to create a new empty frozenset. Return the new set on success or *NULL* on failure. Raise `TypeError` if *iterable* is not actually iterable.

The following functions and macros are available for instances of `set` or `frozenset` or instances of their subtypes.

Py_ssize_t **PySet_Size**(PyObject **anyset*)

Return the length of a `set` or `frozenset` object. Equivalent to `len(anyset)`. Raises a `PyExc_SystemError` if *anyset* is not a `set`, `frozenset`, or an instance of a subtype.

Py_ssize_t **PySet_GET_SIZE**(PyObject **anyset*)

Macro form of `PySet_Size()` without error checking.

int **PySet_Contains**(PyObject **anyset*, PyObject **key*)

Return `1` if found, `0` if not found, and `-1` if an error is encountered. Unlike the Python `__contains__()` method, this function does not automatically convert unhashable sets into temporary frozensets. Raise a `TypeError` if the *key* is unhashable. Raise `PyExc_SystemError` if *anyset* is not a `set`, `frozenset`, or an instance of a subtype.

int **PySet_Add**(PyObject **set*, PyObject **key*)

Add *key* to a `set` instance. Also works with `frozenset` instances (like `PyTuple_SetItem()` it can be used to fill-in the values of brand new frozensets before they are exposed to other code). Return `0` on success or `-1` on failure. Raise a `TypeError` if the *key* is unhashable. Raise a `MemoryError` if there is no room to grow. Raise a `SystemError` if *set* is not an instance of `set` or its subtype.

The following functions are available for instances of `set` or its subtypes but not for instances of `frozenset` or its subtypes.

int **PySet_Discard**(PyObject **set*, PyObject **key*)

Return `1` if found and removed, `0` if not found (no action taken), and `-1` if an error is encountered. Does not raise `KeyError` for missing keys. Raise a `TypeError` if the *key* is unhashable. Unlike the Python `discard()` method, this function does not automatically convert unhashable sets into temporary frozensets. Raise `PyExc_SystemError` if *set* is not an instance of `set` or its subtype.

PyObject* **PySet_Pop**(PyObject **set*)

Return value: New reference.

Return a new reference to an arbitrary object in the *set*, and removes the object from the *set*. Return *NULL* on failure. Raise `KeyError` if the set is empty. Raise a `SystemError` if *set* is not an instance of `set` or its subtype.

int **PySet_Clear**(PyObject **set*)

Empty an existing set of all elements.

int **PySet_ClearFreeList**()

Clear the free list. Return the total number of freed items.

New in version 3.3.