array2d

Efficient general-purpose 2D array.

```
## Example

from array2d import array2d

a = array2d(3, 4, default=0)

a[1, 2] = 5
print(a[1, 2]) # 5
```

Source code:

```
from typing import Callable, Any, Generic, TypeVar, Literal, overload,
Iterator
from linalg import vec2i

T = TypeVar('T')

Neighborhood = Literal['Moore', 'von Neumann']

class array2d(Generic[T]):
    @property
    def n_cols(self) -> int: ...
    @property
    def n_rows(self) -> int: ...
    @property
    def width(self) -> int: ...
    @property
    def height(self) -> int: ...
    @property
    def numel(self) -> int: ...

    def __new__(cls, n_cols: int, n_rows: int, default=None): ...
    def __len__(self) -> int: ...
    def __repr__(self) -> str: ...
    def __iter__(self) -> Iterator[tuple[int, int, T]]: ...

    @overload
    def is_valid(self, col: int, row: int) -> bool: ...
    @overload
    def is_valid(self, pos: vec2i) -> bool: ...

    def get(self, col: int, row: int, default=None) -> T | None:
        """Returns the value at the given position or the default value if
out of bounds."""
    def unsafe_get(self, col: int, row: int) -> T:
        """Returns the value at the given position without bounds
checking."""
    def unsafe_set(self, col: int, row: int, value: T):
        """Sets the value at the given position without bounds checking."""

    @overload
    def __getitem__(self, index: tuple[int, int]) -> T: ...
    @overload
    def __getitem__(self, index: vec2i) -> T: ...
    @overload
    def __getitem__(self, index: tuple[slice, slice]) -> 'array2d[T]': ...
    @overload
    def __setitem__(self, index: tuple[int, int], value: T): ...
    @overload
    def __setitem__(self, index: vec2i, value: T): ...
    @overload
    def __setitem__(self, index: tuple[slice, slice], value: int | float |
str | bool | None | 'array2d[T]'): ...

    def map(self, f: Callable[[T], Any]) -> 'array2d': ...
    def copy(self) -> 'array2d[T]': ...

    def fill_(self, value: T) -> None: ...
    def apply_(self, f: Callable[[T], T]) -> None: ...
    def copy_(self, other: 'array2d[T] | list[T]') -> None: ...

    def tolist(self) -> list[list[T]]: ...
    def render(self) -> str: ...

    # algorithms
    def count(self, value: T) -> int:
        """Counts the number of cells with the given value."""

    def count_neighbors(self, value: T, neighborhood: Neighborhood) ->
'array2d[int]':
        """Counts the number of neighbors with the given value for each
cell."""

    def find_bounding_rect(self, value: T) -> tuple[int, int, int, int]:
        """Finds the bounding rectangle of the given value.
```

Returns a tuple `(x, y, width, height)` or raise `ValueError` if the value is not found.

bisect

`bisect.bisect_left(a, x)`

Return the index where to insert item `x` in list `a`, assuming `a` is sorted.

`bisect.bisect_right(a, x)`

Return the index where to insert item `x` in list `a`, assuming `a` is sorted.

`bisect.insort_left(a, x)`

Insert item `x` in list `a`, and keep it sorted assuming `a` is sorted.
If x is already in a, insert it to the left of the leftmost x.

`bisect.insort_right(a, x)`

Insert item `x` in list `a`, and keep it sorted assuming `a` is sorted.
If x is already in a, insert it to the right of the rightmost x.

Source code:

cmath

Mathematical functions for complex numbers.
https://docs.python.org/3/library/cmath.html

Source code

collections

`collections.Counter(iterable)`

Return a `dict` containing the counts of each element in `iterable`.

`collections.deque`

A double-ended queue.

`collections.defaultdict`

A dictionary that returns a default value when a key is not found.

Source code

dataclasses

`dataclasses.dataclass`

A decorator that is used to add special method to classes, including `__init__`, `__repr__` and `__eq__`.

`dataclasses.asdict(obj) -> dict`

Convert a dataclass instance to a dictionary.

datetime

`datetime.now()`

Returns the current date and time as a `datetime` object.

`date.today()`

Returns the current local date as a `date` object.

Source code

easing

Python wrapper for easing functions.

- `easing.Linear(t: float) -> float`
- `easing.InSine(t: float) -> float`
- `easing.OutSine(t: float) -> float`
- `easing.InOutSine(t: float) -> float`
- `easing.InQuad(t: float) -> float`
- `easing.OutQuad(t: float) -> float`
- `easing.InOutQuad(t: float) -> float`
- `easing.InCubic(t: float) -> float`
- `easing.OutCubic(t: float) -> float`
- `easing.InOutCubic(t: float) -> float`
- `easing.InQuart(t: float) -> float`
- `easing.OutQuart(t: float) -> float`
- `easing.InOutQuart(t: float) -> float`
- `easing.InQuint(t: float) -> float`
- `easing.OutQuint(t: float) -> float`
- `easing.InOutQuint(t: float) -> float`
- `easing.InExpo(t: float) -> float`
- `easing.OutExpo(t: float) -> float`
- `easing.InOutExpo(t: float) -> float`
- `easing.InCirc(t: float) -> float`
- `easing.OutCirc(t: float) -> float`
- `easing.InOutCirc(t: float) -> float`
- `easing.InBack(t: float) -> float`
- `easing.OutBack(t: float) -> float`
- `easing.InOutBack(t: float) -> float`
- `easing.InElastic(t: float) -> float`
- `easing.OutElastic(t: float) -> float`
- `easing.InOutElastic(t: float) -> float`
- `easing.InBounce(t: float) -> float`
- `easing.OutBounce(t: float) -> float`
- `easing.InOutBounce(t: float) -> float`

enum

`enum.Enum`

Base class for creating enumerated constants.
Example:
```python
from enum import Enum

class Color(Enum):
    RED = 1
    GREEN = 2
    BLUE = 3

print(Color.RED)    # Color.RED
print(Color.RED.name)    # 'RED'
print(Color.RED.value)    # 1
```

functools

`functools.cache`

A decorator that caches a function's return value each time it is called. If called later with the same arguments, the cached value is returned, and not re-evaluated.

`functools.reduce(function, sequence, initial=...)`

Apply a function of two arguments cumulatively to the items of a sequence, from left to right, so as to reduce the sequence to a single value. For example, `functools.reduce(lambda x, y: x+y, [1, 2, 3, 4, 5])` calculates `((((1+2)+3)+4)+5)`. The left argument, `x`, is the accumulated value and the right argument, `y`, is the update value from the sequence. If the optional `initial` is present, it is placed before the items of the sequence in the calculation, and serves as a default when the sequence is empty.

`functools.partial(f, *args, **kwargs)`

Return a new partial object which when called will behave like `f` called with the positional arguments `args` and keyword arguments `kwargs`. If more arguments are supplied to the call, they are appended to `args`. If additional keyword arguments are supplied, they extend and override `kwargs`.

Source code

gc

`gc.collect()`

Invoke the garbage collector.

heapq

`heapq.heappush(heap, item)`

Push the value `item` onto the heap, maintaining the heap invariant.

`heapq.heappop(heap)`

Pop and return the smallest item from the heap, maintaining the heap invariant. If the heap is empty, IndexError is raised. To access the smallest item without popping it, use `heap[0]`.

`heapq.heapify(x)`

Transform list `x` into a heap, in-place, in linear time.

`heapq.heappushpop(heap, item)`

Push `item` on the heap, then pop and return the smallest item from the heap. The combined action runs more efficiently than `heappush()` followed by a separate `heappop()`.

`heapq.heapreplace(heap, item)`

Pop and return the smallest item from the heap, and also push the new item. The heap size doesn’t change. If the heap is empty, IndexError is raised.

Source code

json

`json.loads(data: str)`

Decode a JSON string into a python object.

`json.dumps(obj) -> str`

Encode a python object into a JSON string.

linalg

Provide `mat3x3`, `vec2`, `vec3` and `vec4` types.
This classes adopt `torch`'s naming convention. Methods with _ suffix will modify the instance itself.

source:

math

`math.pi`

3.141592653589793

`math.e`

2.718281828459045

`math.inf`

The `inf`.

`math.nan`

The `nan`.

`math.ceil(x)`

Return the ceiling of `x` as a float, the smallest integer value greater than or equal to `x`.

`math.fabs(x)`

Return the absolute value of `x`.

`math.floor(x)`

Return the floor of `x` as a float, the largest integer value less than or equal to `x`.

`math.fsum(iterable)`

Return an accurate floating point sum of values in the iterable. Avoids loss of precision by tracking multiple intermediate partial sums:

```
>>> sum([0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1])
0.9999999999999999
>>> fsum([0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1])
1.0
```

`math.gcd(a, b)`

Return the greatest common divisor of the integers `a` and `b`.

`math.isfinite(x)`

Return `True` if `x` is neither an infinity nor a NaN, and `False` otherwise.

`math.isinf(x)`

Return `True` if `x` is a positive or negative infinity, and `False` otherwise.

`math.isnan(x)`

Return `True` if `x` is a NaN (not a number), and `False` otherwise.

`math.isclose(a, b)`

Return `True` if the values `a` and `b` are close to each other and `False` otherwise.

`math.exp(x)`

Return `e` raised to the power of `x`.

`math.log(x)`

Return the natural logarithm of `x` (to base `e`).

`math.log2(x)`

Return the base-2 logarithm of `x`. This is usually more accurate than `log(x, 2)`.

`math.log10(x)`

Return the base-10 logarithm of `x`. This is usually more accurate than `log(x, 10)`.

`math.pow(x, y)`

Return `x` raised to the power `y`.

`math.sqrt(x)`

Return the square root of `x`.

`math.acos(x)`

Return the arc cosine of `x`, in radians.

`math.asin(x)`

Return the arc sine of `x`, in radians.

`math.atan(x)`

Return the arc tangent of `x`, in radians.

`math.atan2(y, x)`

Return `atan(y / x)`, in radians. The result is between `-pi` and `pi`. The vector in the plane from the origin to point `(x, y)` makes this angle with the positive X axis. The point of `atan2()` is that the signs of both inputs are known to it, so it can compute the correct quadrant for the angle. For example, `atan(1)` and `atan2(1, 1)` are both `pi/4`, but `atan2(-1, -1)` is `-3*pi/4`.

`math.cos(x)`

Return the cosine of `x` radians.

`math.sin(x)`

Return the sine of `x` radians.

`math.tan(x)`

Return the tangent of `x` radians.

`math.degrees(x)`

Convert angle `x` from radians to degrees.

`math.radians(x)`

Convert angle `x` from degrees to radians.

`math.modf(x)`

Return the fractional and integer parts of `x`. Both results carry the sign of `x` and are floats.

`math.factorial(x)`

Return `x` factorial as an integer.

operator

The operator module exports a set of efficient functions corresponding to the intrinsic operators of Python. For example, `operator.add(x, y)` is equivalent to the expression `x+y`. Many function names are those used for special methods, without the double underscores.

Mapping Operators to Functions

Operation	Syntax	Function
Ordering	`a <= b`	`le(a, b)`
Ordering	`a < b`	`lt(a, b)`
Ordering	`a >= b`	`ge(a, b)`
Ordering	`a > b`	`gt(a, b)`
Equality	`a == b`	`eq(a, b)`
Equality	`a != b`	`ne(a, b)`
Bitwise AND	`a & b`	`and_(a, b)`
Bitwise OR	`a | b`	`or_(a, b)`
Bitwise XOR	`a ^ b`	`xor(a, b)`
Bitwise Inversion	`~a`	`invert(a)`
Left Shift	`a << b`	`lshift(a, b)`
Right Shift	`a >> b`	`rshift(a, b)`
Identity	`a is b`	`is_(a, b)`
Identity	`a is not b`	`is_not(a, b)`
Negation (Logical)	`not a`	`not_(a)`
Negation (Arithmetic)	`-a`	`neg(a)`
Truth Test	`bool(a)`	`truth(a)`
Containment Test	`b in a`	`contains(a, b)`
Addition	`a + b`	`add(a, b)`
Subtraction	`a - b`	`sub(a, b)`
Multiplication	`a * b`	`mul(a, b)`
Division	`a / b`	`truediv(a, b)`
Division	`a // b`	`floordiv(a, b)`
Modulo	`a % b`	`mod(a, b)`
Exponentiation	`a ** b`	`pow(a, b)`
Matrix Multiplication	`a @ b`	`matmul(a, b)`
Indexing	`a[b]`	`getitem(a, b)`
Index Assignment	`a[b] = c`	`setitem(a, b, c)`
Index Deletion	`del a[b]`	`delitem(a, b)`

In-place Operators

Operation	Syntax	Function
Addition	`a += b`	`iadd(a, b)`
Subtraction	`a -= b`	`isub(a, b)`
Multiplication	`a *= b`	`imul(a, b)`
Division	`a /= b`	`itruediv(a, b)`
Division	`a //= b`	`ifloordiv(a, b)`
Modulo	`a %= b`	`imod(a, b)`
Bitwise AND	`a &= b`	`iand(a, b)`
Bitwise OR	`a |= b`	`ior(a, b)`
Bitwise XOR	`a ^= b`	`ixor(a, b)`
Left Shift	`a <<= b`	`ilshift(a, b)`
Right Shift	`a >>= b`	`irshift(a, b)`

os

Contains system functions for work with files and folders.

NOTICE: When using using fileio functions files can only be created and read from the user profile folder that is used for macros. Any given file path is considered relative to this folder.
Also, using/changing current working dir is not recommended since it is defined for a process and not per thread. Macros can run in different threads of the process. Therefore it is recommended to use absolute paths.

`os.getcwd()`

Returns the current working directory as a string.

`os.chdir(path: str)`

Changes the current working directory to the given path.

`os.remove(path: str)`

Removes the file at the given path.

`os.pathexists(path: str)`

Check if the given path exists. Parameter **path** can be a file path in which case existance of the file is checked. It also can be a folder path in which case existance of the folder is checked. Returns bool, True or False.

pickle

`pickle.dumps(obj) -> bytes`

Return the pickled representation of an object as a bytes object.

`pickle.loads(b: bytes)`

Return the unpickled object from a bytes object.

What can be pickled and unpickled?

The following types can be pickled:

- [x] None, True, and False;
- [x]integers, floating-point numbers;
- [x] strings, bytes;
- [x] tuples, lists, sets, and dictionaries containing only picklable objects;
- [ ] functions (built-in and user-defined) accessible from the top level of a module (using def, not lambda);
- [x]classes accessible from the top level of a module;
- [x] instances of such classes
- `[x]PY_STRUCT_LIKE` objects

The following magic methods are available:

- `[x] __getnewargs__`
- `[ ] __getstate__`
- `[ ] __setstate__`
- `[ ] __reduce__`

random

`random.seed(a)`

Set the random seed.

`random.random()`

Return a random float number in the range [0.0, 1.0).

`random.randint(a, b)`

Return a random integer in the range [a, b].

`random.uniform(a, b)`

Return a random float number in the range [a, b).

`random.choice(seq)`

Return a random element from a sequence.

`random.shuffle(seq)`

Shuffle a sequence inplace.

`random.choices(population, weights=None, k=1)`

Return a k sized list of elements chosen from the population with replacement.

sys

`sys.version`

The version of pkpy.

`sys.platform`

May be one of:

- `win32`
- `linux`
- `darwin`
- `android`
- `ios`
- `emscripten`

`sys.argv`

The command line arguments. Set by `vm->set_main_argv`.

time

`time.time()`

Returns the current time in seconds since the epoch as a floating point number.

`time.sleep(secs)`

Suspend execution of the calling thread for the given number of seconds.

`time.localtime()`

Returns the current struct time as a `struct_time`object.

traceback

`traceback.print_exc() -> None`

Print the last exception and its traceback.

`traceback.format_exc() -> str`

Return the last exception and its traceback as a string.

typing

Placeholder module for type hints.

source code:

```
class _Placeholder:
    def __init__(self, *args, **kwargs):
        pass
    def __getitem__(self, *args):
        return self
    def __call__(self, *args, **kwargs):
        return self
    def __and__(self, other):
        return self
    def __or__(self, other):
        return self
    def __xor__(self, other):
        return self


_PLACEHOLDER = _Placeholder()

List = _PLACEHOLDER
Dict = _PLACEHOLDER
Tuple = _PLACEHOLDER
Set = _PLACEHOLDER
Any = _PLACEHOLDER
Union = _PLACEHOLDER
Optional = _PLACEHOLDER
Callable = _PLACEHOLDER
Type = _PLACEHOLDER
Protocol = _PLACEHOLDER

Literal = _PLACEHOLDER
LiteralString = _PLACEHOLDER

Iterable = _PLACEHOLDER
Generator = _PLACEHOLDER
```

```
Hashable = _PLACEHOLDER

TypeVar = _PLACEHOLDER
Self = _PLACEHOLDER

class Generic:
    pass

TYPE_CHECKING = False

# decorators
overload = lambda x: x
final = lambda x: x
```

fileio

Extension module contains functions for working with files.

NOTICE: Files can only be created and read from the user profile folder that is used for macros. Any given file path is considered relative to this folder.

`fileio.open(fname: str, mode: str) -> File`

Open the file with given name. Mode string specifies access mode. Returns a **File** object.
mode can be:
"r": Open for reading. The file must exist.
"w": Open for writing. Creates an empty file or truncates an existing file.
"a": Open for appending. Writes data at the end of the file. Creates the file if it does not exist.
"r+": Open for reading and writing. The file must exist.
"w+": Open for reading and writing. Creates an empty file or truncates an existing file.
"a+": Open for reading and appending. The file is created if it does not exist.

`File.close()`

Close previously opened file.

`File.read(bytes:int) -> str   #-1 all`

Read data from the opened file. Returns a string that is read.
If bytes is specified and positive, then this number of bytes is read. Otherwise full content of the file is read.

`File.write(text: str)`

Write a string to the file.

`File.writeline(text: str)`

Write a text line to the file.

`File.writelines(lines: list)`

Write multiple lines of text to the file. A newline character is NOT appended at the end of each line.

`File.readline(maxbytes: int)-> str  #-1  full line`

Reads a line of text from the opened file. Maximum of **maxbytes** is read if **maxbytes** is specified and if it is a positive number. Otherwise a full line of text is read up do the newline character. A string is returned.
A newline character is NOT removed from the end of line.

`File.seek(offset: int, mode: int)`

Seeks in the file given an offset and the mode.
offset − This is the number of positions (characters/bytes) of the read/write pointer to move within the file.
mode − (Optional) It defaults to 0;
0 = absolute file positioning,
1 = seek relative to the current position,
2 = seek relative to the file's end.

`File.tell() -> int`

Returns an integer that is a file object's current position from the beginning of the file in the form of bytes.

`File.flush()`

Flushes the output buffer for the file. When data is written to a disk file, the data is typically not written directly to the disk on every write operation. Instead, for performance reasons, data is maintained in the buffer and saved only when certain data amount is gathered or some other criteria is met.
Using flush() function forces emptying output buffer and thus sends gathered data to the OS for writing to a physical drive.