```
# builin math
```

```
import math
```

```
math.sqrt(10)
```

```
3.1622776601683795
```

```
math.ceil(4.5) # finds the nearest large integer
```

```
5
```

```
math.floor(4.5) # finds the nearest small integer
```

```
4
```

```
math.pi
```

```
3.141592653589793
```

```
import mymath # create custom my math module and use it
```

```
### import sample_data.mymath # create custom my math module an
```

```
mymath.pi
```

```
100
```

```
mymath.add(2,3)
```

```
5
```

```
mymath.add(2,3,4)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-4f9bef5fdacf> in <cell line: 1>()
----> 1 mymath.add(2,3,4)

TypeError: add() takes 2 positional arguments but 3 were given

SEARCH STACK OVERFLOW
```

```
help(math)
```

```
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

        The result is between 0 and pi.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.

        The result is between -pi/2 and pi/2.
```

```
asinh(x, /)
    Return the inverse hyperbolic sine of x.

atan(x, /)
    Return the arc tangent (measured in radians) of x.

    The result is between -pi/2 and pi/2.

atan2(y, x, /)
    Return the arc tangent (measured in radians) of y/x.

    Unlike atan(y/x), the signs of both x and y are considered.

atanh(x, /)
    Return the inverse hyperbolic tangent of x.

ceil(x, /)
    Return the ceiling of x as an Integral.

    This is the smallest integer >= x.

comb(n, k, /)
    Number of ways to choose k items from n items without repetition and without order.

    Evaluates to n! / (k! * (n - k)!) when k <= n and evaluates
    to zero when k > n.

    Also called the binomial coefficient because it is equivalent
    to the coefficient of k-th term in polynomial expansion of the
    expression (1 + x)**n.

    Raises TypeError if either of the arguments are not integers.
    Raises ValueError if either of the arguments are negative.
```

```
help(math.floor)
```

```
Help on built-in function floor in module math:

floor(x, /)
    Return the floor of x as an Integral.

    This is the largest integer <= x.
```

```
import math
```

```
math.prod([1,2,3])
```

```
6
```

HW: can we write all the content of a module to another file?: read the module and copy content to a new file

method 1 of importing

```
import math # whole module
```

```
math.pi
```

```
3.141592653589793
```

method 2 of importing

```
from math import * # we are importing all the members of math m
```

```
ceil(1.3)
```

```
2
```

```
sqrt(5)
```

```
2.23606797749979
```

```
pi
```

```
3.141592653589793
```

- disadvantage:

1. Imports everything into current module
2. Overwrites in case of same member present across different modules

Method 3

```
from math import pi, sqrt
```

```
pi
```

```
3.141592653589793
```

```
sqrt(100)
```

```
10.0
```

```
from functools import reduce
```

```
import functools
```

```
help(functools)
```

```
        function (defaults to functools.WRAPPER_UPDATES)

    wraps(wrapped, assigned=('__module__', '__name__', '__qualname__', '__doc__', '__annotations__'), updated=('__dict__'
        Decorator factory to apply update_wrapper() to a wrapper function

        Returns a decorator that invokes update_wrapper() with the decorated
        function as the wrapper argument and the arguments to wraps() as the
        remaining arguments. Default arguments are as for update_wrapper().
        This is a convenience function to simplify applying partial() to
        update_wrapper().

DATA
    WRAPPER_ASSIGNMENTS = ('__module__', '__name__', '__qualname__', '__do...
    WRAPPER_UPDATES = ('__dict__',)
    __all__ = ['update_wrapper', 'wraps', 'WRAPPER_ASSIGNMENTS', 'WRAPPER_...

FILE
    /usr/lib/python3.9/functools.py
```

- Aliases

```
from math import sqrt as r # alias to a function inside a modul

r(100)
```

```
10.0
```

```
import math as m # alias to module

m.ceil(10.1)
```

```
11
```

```
from mymath import pi as mypi
from math import pi as pi

print(mypi)
print(pi)
```

```
100
3.141592653589793
```

```
import numpy as np
```

- ***HOMEWORK***:

Create a custom math module with 4 functions -

- add
- sub
- divide
- multiply

import all functions using different methods

```
import numpy # package and library

help(numpy)
```

```
Help on package numpy:

NAME
    numpy

DESCRIPTION
```

```
NumPy
=====

Provides
  1. An array object of arbitrary homogeneous items
  2. Fast mathematical operations over arrays
  3. Linear Algebra, Fourier Transforms, Random Number Generation

How to use the documentation
----------------------------
Documentation is available in two forms: docstrings provided
with the code, and a loose standing reference guide, available from
`the NumPy homepage <https://www.scipy.org>`_.

We recommend exploring the docstrings using
`IPython <https://ipython.org>`_, an advanced Python shell with
TAB-completion and introspection capabilities.  See below for further
instructions.

The docstring examples assume that `numpy` has been imported as `np`::

  >>> import numpy as np

Code snippets are indicated by three greater-than signs::

  >>> x = 42
  >>> x = x + 1

Use the built-in ``help`` function to view a function's docstring::

  >>> help(np.sort)
  ... # doctest: +SKIP

For some objects, ``np.info(obj)`` may provide additional help.  This is
particularly true if you see the line "Help on ufunc object:" at the top
of the help() page.  Ufuncs are implemented in C, not Python, for speed.
The native Python help() does not know how to view their help, but our
np.info() function does.

To search for documents containing a keyword, do::

  >>> np.lookfor('keyword')
  ... # doctest: +SKIP

General-purpose documents like a glossary and help on the basic concepts
of numpy are available under the ``doc`` sub-module::

  >>> from numpy import doc
  >>> help(doc)
  ... # doctest: +SKIP

Available subpackages
```

ls /usr/local/lib/python3.9/dist-packages/numpy

```
array_api/            fft/                  polynomial/
compat/               _globals.py           __pycache__/
__config__.py         __init__.cython-30.pxd _pytesttester.py
conftest.py           __init__.pxd          _pytesttester.pyi
core/                 __init__.py           py.typed
ctypeslib.py          __init__.pyi          random/
ctypeslib.pyi         lib/                  setup.py
_distributor_init.py  LICENSE.txt           testing/
distutils/            linalg/               tests/
doc/                  ma/                   typing/
dual.py               matlib.py             _version.py
f2py/                 matrixlib/            version.py
```

ls /usr/local/lib/python3.9/dist-packages/numpy/polynomial

```
chebyshev.py   hermite.pyi   legendre.py    polynomial.pyi  tests/
chebyshev.pyi  __init__.py   legendre.pyi   polyutils.py
hermite_e.py   __init__.pyi  _polybase.py   polyutils.pyi
hermite_e.pyi  laguerre.py   _polybase.pyi  __pycache__/
hermite.py     laguerre.pyi  polynomial.py  setup.py
```

```
import os
# which what is present inside a folder
os.listdir() # windows
```

```
ls # mac and linux
```

```
from numpy # imported numpy library

from numpy import polynomial # imported polynomial pacakge
```

```
  File "<ipython-input-45-b3127725f678>", line 1
    }
    ^
SyntaxError: unmatched '}'
```

SEARCH STACK OVERFLOW

```
help(polynomial)
```

```
Help on package numpy.polynomial in numpy:

NAME
    numpy.polynomial - A sub-package for efficiently dealing with polynomials.

DESCRIPTION
    Within the documentation for this sub-package, a "finite power series,"
    i.e., a polynomial (also referred to simply as a "series") is represented
    by a 1-D numpy array of the polynomial's coefficients, ordered from lowest
    order term to highest.  For example, array([1,2,3]) represents
    ``P_0 + 2*P_1 + 3*P_2``, where P_n is the n-th order basis polynomial
    applicable to the specific module in question, e.g., `polynomial` (which
    "wraps" the "standard" basis) or `chebyshev`.  For optimal performance,
    all operations on polynomials, including evaluation at an argument, are
    implemented as operations on the coefficients.  Additional (module-specific)
    information can be found in the docstring for the module of interest.

    This package provides *convenience classes* for each of six different kinds
    of polynomials:

            ========================    ================
            **Name**                    **Provides**
            ========================    ================
            `~polynomial.Polynomial`    Power series
            `~chebyshev.Chebyshev`      Chebyshev series
            `~legendre.Legendre`        Legendre series
            `~laguerre.Laguerre`        Laguerre series
            `~hermite.Hermite`          Hermite series
            `~hermite_e.HermiteE`       HermiteE series
            ========================    ================

    These *convenience classes* provide a consistent interface for creating,
    manipulating, and fitting data with polynomials of different bases.
    The convenience classes are the preferred interface for the `~numpy.polynomial`
    package, and are available from the ``numpy.polynomial`` namespace.
    This eliminates the need to navigate to the corresponding submodules, e.g.
    ``np.polynomial.Polynomial`` or ``np.polynomial.Chebyshev`` instead of
    ``np.polynomial.polynomial.Polynomial`` or
    ``np.polynomial.chebyshev.Chebyshev``, respectively.
    The classes provide a more consistent and concise interface than the
    type-specific functions defined in the submodules for each type of polynomial.
    For example, to fit a Chebyshev polynomial with degree ``1`` to data given
    by arrays ``xdata`` and ``ydata``, the
    `~chebyshev.Chebyshev.fit` class method::

        >>> from numpy.polynomial import Chebyshev
        >>> c = Chebyshev.fit(xdata, ydata, deg=1)

    is preferred over the `chebyshev.chebfit` function from the
    ``np.polynomial.chebyshev`` module::

        >>> from numpy.polynomial.chebyshev import chebfit
        >>> c = chebfit(xdata, ydata, deg=1)

    See :doc:`routines.polynomials.classes` for more details.

    Convenience Classes
    ===================
```

```
# from custompackage import mymath
```

```
import random
random.randint(0,10)
```

```
4
```

```
random.randint(0,10)
```

```
5
```

```
random.seed(100)
print(random.randint(0,10))
print(random.randint(0,10))
print(random.randint(0,10))
print(random.randint(0,10))
print(random.randint(0,10))
```

```
2
7
7
2
6
```

```
random.seed(100)
print(random.randint(0,10))
print(random.randint(0,10))
print(random.randint(0,10))
print(random.randint(0,10))
print(random.randint(0,10))
```

```
2
7
7
2
6
```

```
import arithmetic

def calculate(number):
    return number - 2

print(calculate(1)) # main or aritmatic: Main : n-2 :1-2= -1
print(arithmetic.calculate(1)) # main or aritmatic: aritmatic:


def calculate(number):
    return number - 2

from arithmetic import *  # calculate has n+2

print(calculate(1)) # main or aritmatic: aritmatic : n-2 :1+2=
print(calculate(1)) # main or aritmatic: aritmatic: n+2: 1+2= 3
```

Exception Handling

```
def func1():
    pass

def func2():
    pass

func1() # error in func1
func2()
```

Division by zero

```
def something(x):
    print(1 / x)
    print("A")

something(0)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-56-f1041be5548b> in <cell line: 5>()
      3     print("A")
      4
----> 5 something(0)

<ipython-input-56-f1041be5548b> in something(x)
      1 def something(x):
----> 2     print(1 / x)
      3     print("A")
      4
      5 something(0)

ZeroDivisionError: division by zero
```

SEARCH STACK OVERFLOW

```
print(not_defined_a)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-57-fb1798268bb1> in <cell line: 1>()
----> 1 print(not_defined_a)

NameError: name 'not_defined_a' is not defined
```

SEARCH STACK OVERFLOW

```
if 573:

print("Hello")
```

```
  File "<ipython-input-58-d77ca14b3f5b>", line 3
    print("Hello")
    ^
IndentationError: expected an indented block
```

SEARCH STACK OVERFLOW

```
import arithmatic
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-59-c4b614e75feb> in <cell line: 1>()
----> 1 import arithmatic

ModuleNotFoundError: No module named 'arithmatic'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.
```

dir(__builtins__)

```
['ArithmeticError',
 'AssertionError',
 'AttributeError',
 'BaseException',
 'BlockingIOError',
 'BrokenPipeError',
 'BufferError',
 'BytesWarning',
 'ChildProcessError',
 'ConnectionAbortedError',
 'ConnectionError',
 'ConnectionRefusedError',
 'ConnectionResetError',
 'DeprecationWarning',
 'EOFError',
 'Ellipsis',
 'EnvironmentError',
 'Exception',
 'False',
 'FileExistsError',
 'FileNotFoundError',
 'FloatingPointError',
 'FutureWarning',
 'GeneratorExit',
 'IOError',
 'ImportError',
 'ImportWarning',
 'IndentationError',
 'IndexError',
 'InterruptedError',
 'IsADirectoryError',
 'KeyError',
 'KeyboardInterrupt',
 'LookupError',
 'MemoryError',
 'ModuleNotFoundError',
 'NameError',
 'None',
 'NotADirectoryError',
 'NotImplemented',
 'NotImplementedError',
 'OSError',
 'OverflowError',
 'PendingDeprecationWarning',
 'PermissionError',
 'ProcessLookupError',
 'RecursionError',
 'ReferenceError',
 'ResourceWarning',
 'RuntimeError',
 'RuntimeWarning',
 'StopAsyncIteration',
 'StopIteration',
 'SyntaxError',
 'SyntaxWarning',
 'SystemError',
 'SystemExit',
 'TabError',
```

```
x= 0
try:
    # any code
    print(1/x)

except: # what happens if you encounter an error
    print("x value can't to be Zero")
```

```
print("hello")
```

```
x value can't to be Zero
hello
```

```
x= 0
try:
    # any code
    print(1/x)

except: # what happens if you encounter an error
    print("x value can't to be Zero")
```

```
x value can't to be Zero
```

```
x= 0
# any code
print(1/x)
print("hello")
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-66-23f00c7f83a4> in <cell line: 3>()
      1 x= 0
      2 # any code
----> 3 print(1/x)
      4 print("hello")

ZeroDivisionError: division by zero

SEARCH STACK OVERFLOW
```

```
x= []
try:
    # any code
    print(1/x)

except: # what happens if you encounter an error # generic exce
    print("Some error Occured")
```

```
x value can't to be Zero
```

```
x= 0
try:
    # any code
    print(1/x) # line1

except ZeroDivisionError: # specialized except block
    print("X can't be Zero") #line 2

except: # what happens if you encounter an error # generic exce
    print("Some error Occured")  # line3
```

```
X can't be Zero
```

```
1/"A"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-70-5c7edeb1a2cd> in <cell line: 1>()
----> 1 1/"A"

TypeError: unsupported operand type(s) for /: 'int' and 'str'

SEARCH STACK OVERFLOW
```

```
x= "A"
try:
    # any code
    print(1/x) # line1

except ZeroDivisionError: # specialized except block
    print("X can't be Zero") #line 2

except: # what happens if you encounter an error # generic exce
    print("Some error Occured")  # line 3
```

```
Some error Occured
```

```
x= "A"
try:
    # any code
    print(1/x) # line1

except ZeroDivisionError: # specialized except block
    print("X can't be Zero") #line 2

except Exception as e: # what happens if you encounter an error
    print("Some error Occured: ",  str(e))  # line 3
```

```
Some error Occured:  unsupported operand type(s) for /: 'int' and 'str'
```

```
x= int(input("Enter the number")) # "11"
try:
    # any code
    print(1/x) # line1

except ZeroDivisionError: # specialized except block
    print("X can't be Zero") #line 2
except TypeError:
    print("x is of type: ", type("A"), ". Expected integer.") #
except Exception as e: # what happens if you encounter an error
    print("Some error Occured: ",  str(e))  # line 4
```

```
Enter the number11
0.09090909090909091
```

```
# getting what exception is occuring
l1 = [2, 0, "hello", None]

for e in l1:
    try:
        print(f"Current element - {e}")
        result = 5 / int(e)
        print(f"Result - {result}")
    except Exception as ex:
        print(f"Excpetion - {ex}")

    print("-"*25)

print("Execution Successful!")
```

```
Current element - 2
Result - 2.5
-------------------------
Current element - 0
Excpetion - division by zero
-------------------------
Current element - hello
Excpetion - invalid literal for int() with base 10: 'hello'
-------------------------
Current element - None
Excpetion - int() argument must be a string, a bytes-like object or a number, not 'NoneType'
-------------------------
Execution Successful!
```

```
5/"Hello"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-77-7e0286149d2b> in <cell line: 1>()
----> 1 5/"Hello"

TypeError: unsupported operand type(s) for /: 'int' and 'str'
```

SEARCH STACK OVERFLOW

```
5/None
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-78-b6f73012e47f> in <cell line: 1>()
----> 1 5/None

TypeError: unsupported operand type(s) for /: 'int' and 'NoneType'
```

SEARCH STACK OVERFLOW

```
# getting what exception is occuring
l1 = [2, 0, "hello", None]

for e in l1:
    try:
        print(f"Current element - {e}")
        result = 5 / int(e)
        print(f"Result - {result}")
```

```
    except ZeroDivisionError:
        print("e can't be Zero")
    except TypeError:
        print("e must be int")

    except ValueError:
        print("Can't change e to number")
    except Exception as ex:
        print(f"Exception - {ex}")

    print("-"*25)

print("Execution Successful!")
```

```
Current element - 2
Result - 2.5
-------------------------
Current element - 0
e can't be Zero
-------------------------
Current element - hello
Can't change e to number
-------------------------
Current element - None
e must be int
-------------------------
Execution Successful!
```

```
int("hello")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-80-a6f1987f81d0> in <cell line: 1>()
----> 1 int("hello")

ValueError: invalid literal for int() with base 10: 'hello'
```

SEARCH STACK OVERFLOW

```
l1 = [2, 0, "hello", None]
for e in l1:
    try:
        print(f"Current element - {e}")
        result = 5 / int(e)
        print(f"Result - {result}")  # line 1

    except Exception as ex: # this should always be at the bott
        print(f"Defualt Exception - {ex}") # line 2

    except ZeroDivisionError:
        print("e can't be Zero") # line 3

    except TypeError:
        print("e must be int") # line 4

    except ValueError:
        print("Can't change e to number") # line 5
```

```
    print("-"*25)

print("Execution Successful!")
```

```
Current element - 2
Result - 2.5
-------------------------
Current element - 0
Defualt Exception - division by zero
-------------------------
Current element - hello
Defualt Exception - invalid literal for int() with base 10: 'hello'
-------------------------
Current element - None
Defualt Exception - int() argument must be a string, a bytes-like object or a number, not 'NoneType'
-------------------------
Execution Successful!
```

```
#
```

Finally

```
x= 1

try:
    print(1/x) # line 1

except:
    print("Some Error Occured") # line 2

finally:
    print("I get executed irrespective of error occurs or not")
```

```
1.0
I get executed irrespective of error occurs or not
```

```
x= 0

try:
    print(1/x) # line 1

except:
    print("Some Error Occured") # line 2

finally:
    print("I get executed irrespective of error occurs or not")
```

```
Some Error Occured
I get executed irrespective of error occurs or not
```

```
x= 0

try:
    print(1/x) # line 1

except:
    print("Some Error Occured") # line 2
```

```
    try:
        print("Inside Except")

    except:
        print("Excepts except")

    finally:
        print("I get executed irrespective of error occurs or n

finally:
    print("I get executed irrespective of error occurs or not")
```

```
Some Error Occured
Inside Except
I get executed irrespective of error occurs or not
I get executed irrespective of error occurs or not
```

```
  File "<ipython-input-91-56f4c902307c>", line 2
    if
       ^
SyntaxError: invalid syntax
```

SEARCH STACK OVERFLOW

- In the doubt sesssion, i can take creating custom Exceptions.

Double-click (or enter) to edit

```
import  traceback

x= 0

try:
    print(1/x) # line 1

except:
    print("Some Error Occured") # line 2
    print(traceback.format_exc())
```

```
Some Error Occured
Traceback (most recent call last):
  File "<ipython-input-90-e02a42df8be5>", line 6, in <cell line: 5>
    print(1/x) # line 1
ZeroDivisionError: division by zero
```

- Gradient Descent
- BackPropagation

- Creating and raising customException