計算物理概論

Introduction to Computational Physics (PHYS290000)

Lecture 8: Advanced Python (part 2)

Instructor: 潘國全

kuochuan.pan@gapp.nthu.edu.tw

Last week

1. Advanced topics
2. Errors and Exceptions
3. Command line arguments
4. Numerical integral

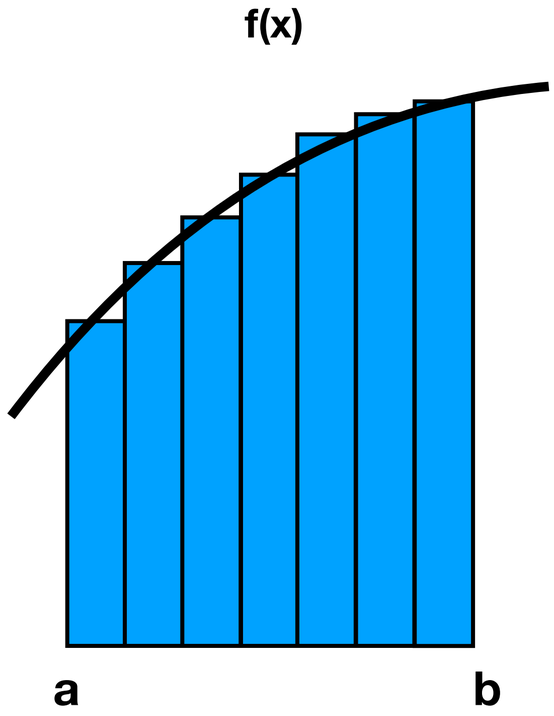
Today’s plan

1. Warm-up (numerical integral)
2. Jupyter notebook
3. Lambda
4. Performance measurement
5. File wildcards (pathlib and glob)
6. NumPy (arrays & random numbers)

Warm-up: Numerical Integral

Write a numerical integrator

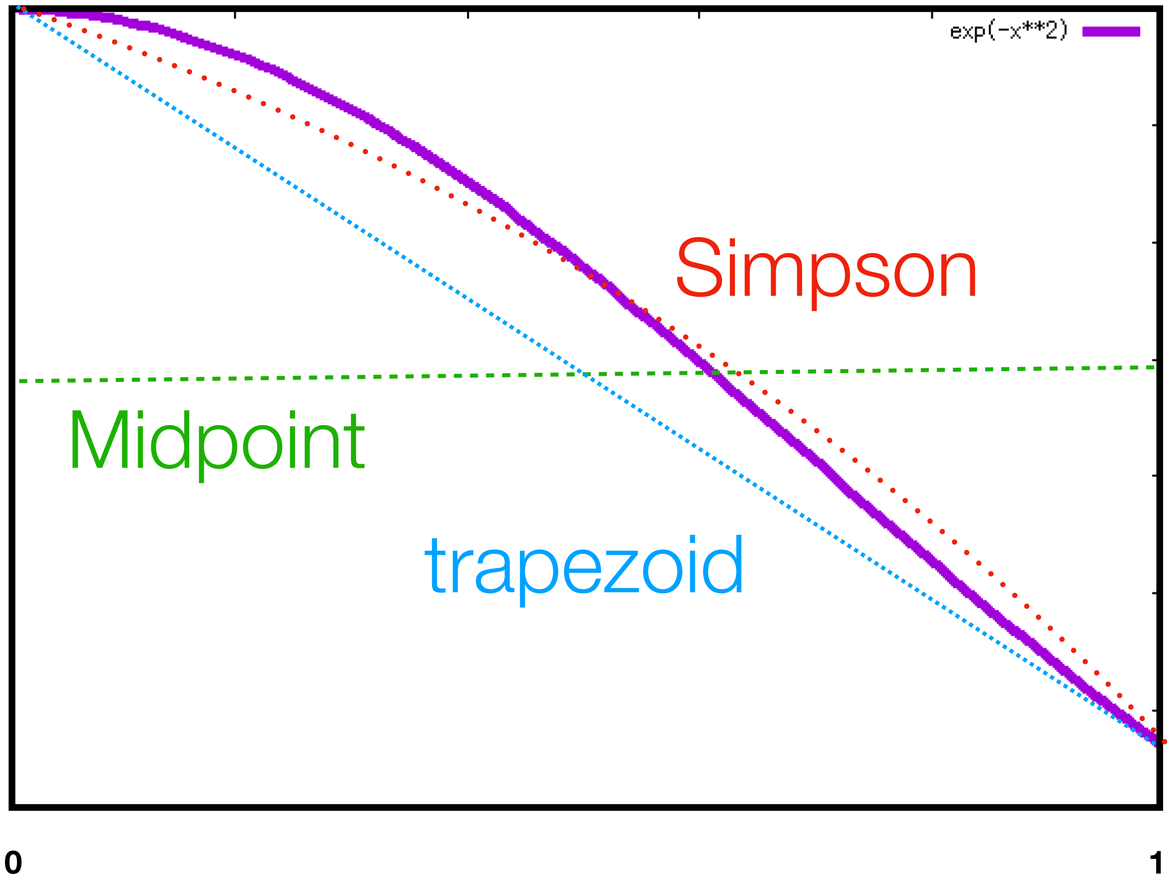
1. To integrate a function from a to b is to evaluate the area under the function in [a,b]



2. Let's write a python class called "Integrator" that can do finite integral of an arbitrary function from a to b with dx = [b-a]/N, where N is the user-defined, number of divisions.

3. To evaluate the area of each sub-devision, we could use "midpot", "trapezoid", or "simpson" methods

Write a numerical integrator



4.

Midpoint rule

$$\int_a^b f(x)dx \sim (b-a)f(\frac{a+b}{2})$$

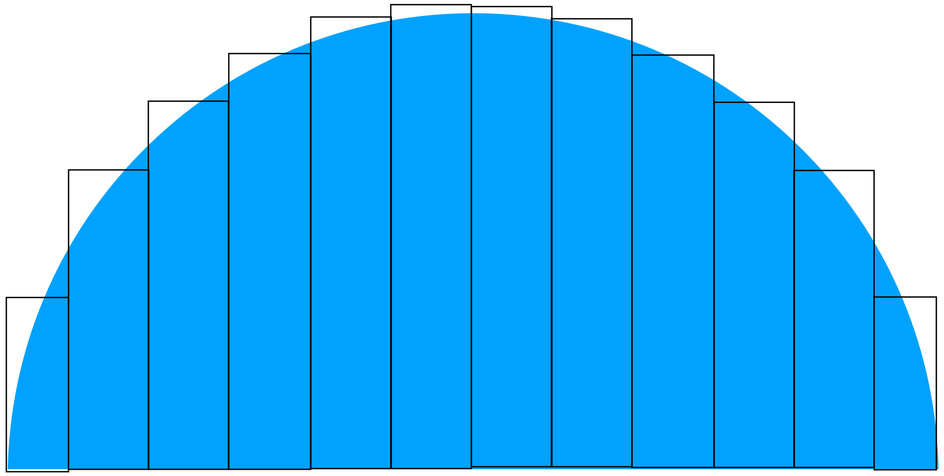
Trapezoidal rule

$$\int_a^b f(x)dx \sim (b-a)(\frac{f(a)+f(b)}{2})$$

Simpson's rule

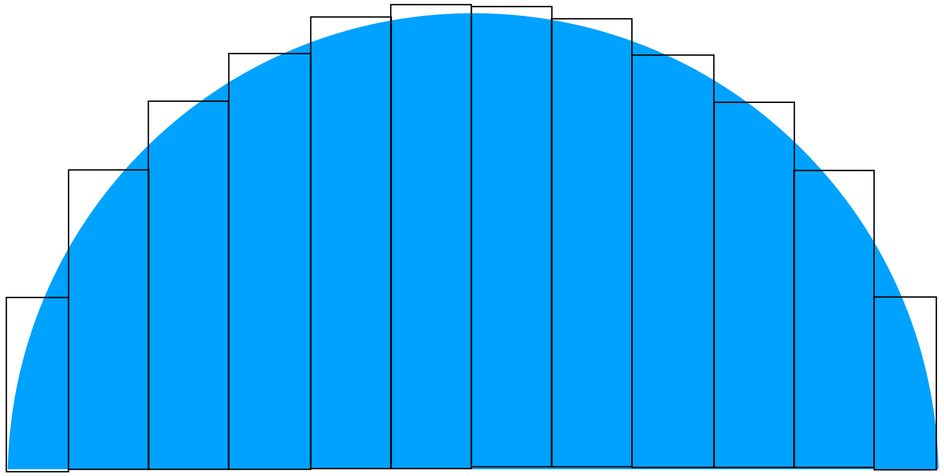
$$\int_a^b f(x)dx \sim \frac{(b-a)}{6}(f(a)+4f(\frac{a+b}{2})+f(b))$$

Example: Calculate Pi



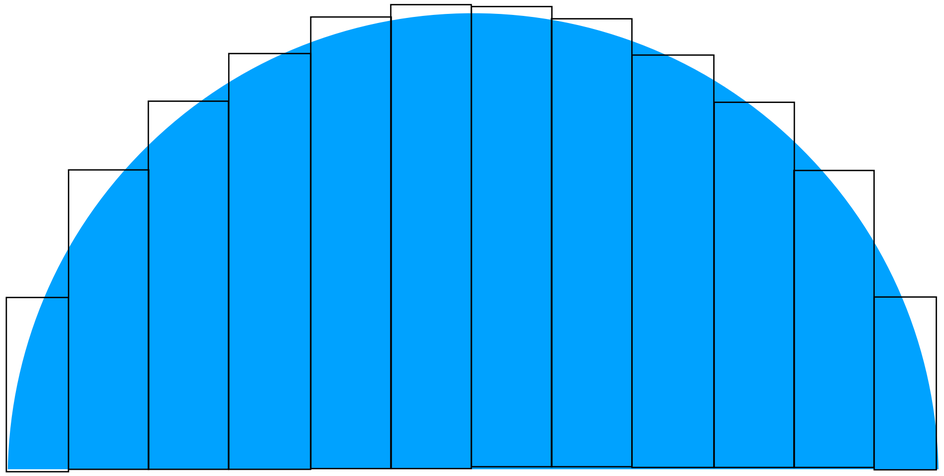
1. The area of a half-unit-circle is equal to pi/2
2. Use the integrator we wrote to evaluate pi
3. Start from the “midpoint” method.
4. See the sample code “integrator_template.py”
5. Note: the program could be slow in this example

Example: Calculate Pi



```python
import math

class Integrator:

    def __init__(self, func):
        """
        setup the integrator with a given function
        """

        # TODO

        return

    def midpoint(self,a=-1,b=1,N=10_000):
        """
        Assume a < b
        """

        # TODO

        return area


if __name__=='__main__':

    def hcirc(x):
        return math.sqrt(1-x**2)

    integrate = Integrator(func=hcirc)
    area = integrate.midpoint(a=-1,b=1,N=10000)
    print(2*area)
```

Exercise: other methods



1. Now, it is your time to practice implementing the trapezoidal rule and Simpson's rule.
2. Write two class methods "trap" and "simpson" to do the same integrals.
3. Try different N, see your results with different methods are less or larger than the true PI.
4. Remember to raise an error if b <= a

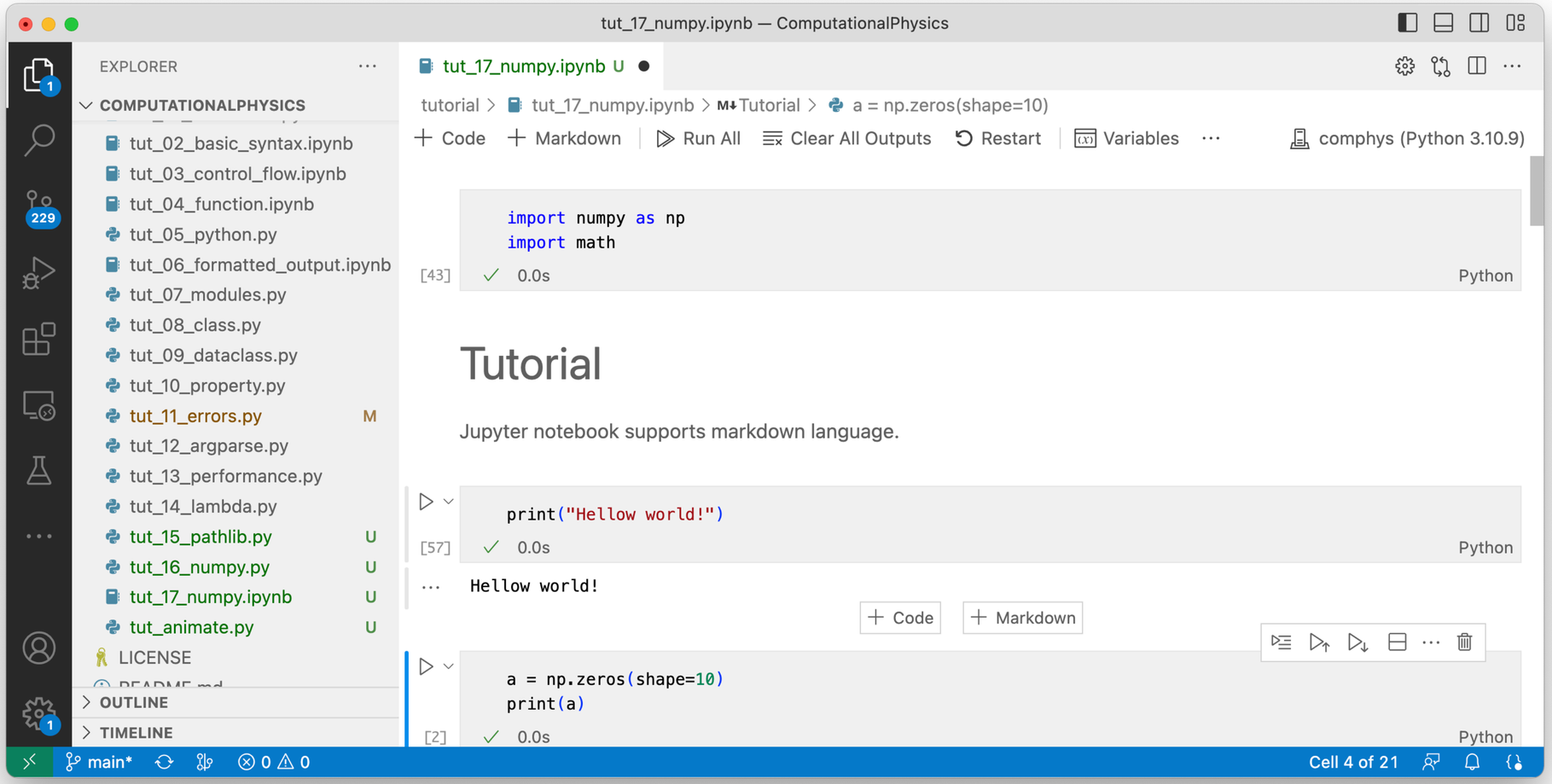
Jupyter notebook

Jupyter notebook

1. File extension is “.ipynb”
2. An interactive computing environment to quickly access programming results
3. Iterative development and faster feedback
4. Rich media integration: markdown, image, video, …etc.
5. Reproducible workflows
6. Disadvantages: (1) version control issues (2) slower than typical python scripts, (3) not ideal for production code (cannot import)

Jupyter notebook

1. (Demo)



Exercise: Jupyter notebook

1. Rewrite and port your numerical integration code in a Jupyter notebook.

2. Alternatively, you could import your Integrator class in a Jupyter notebook

Python Lambda

Write a numerical integrator

1. Python's "lambda" is a very powerful way to define functions.
2. A lambda function is a small anonymous function (no function name)
3. A lambda function can take any number of arguments, but can only have one expression
4. Syntax: `lambda arguments : expression`
5. Example

```
hcirc = lambda x: math.sqrt(1-x**2)
```

Write a numerical integrator

6. More examples:

```
# declare a lambda function
greet = lambda : print("Hello world!")

# call the lambda function
greet()
```

7. Create a list in one line with lambda

```
list1 = [(lambda x: x**2)(x) for x in range(10)]
#or
list2 = [x**2 for x in range(10)]
print(list1)
print(list2)
```

Write a numerical integrator

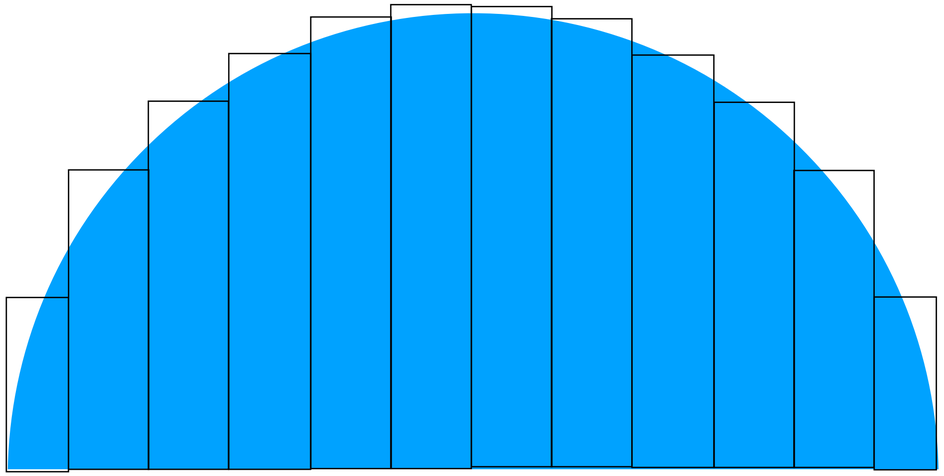
8. Use lambda to map a list

```
a  = [1, 2, 3, 4, 5, 6]
b = list(map(lambda x: x * 2 , a))

#or
c = [x*2 for x in a]
print(b)
print(c)
```

9. Use lambda to filter a list

```
a = [1, 2, 3, 4, 5, 6, 7, 8]
b = list(filter(lambda x: (x%2 == 0) , a))

# or
c = [x for x in a if x%2 == 0]
print(b)
print(c)
```

Exercise: use lambda



1. Back to your Integrator(), now your lambda to define your function
2. Write a new method called midpoint2. Use lambda or list comprehensions to avoid the for loop when summing the area.
3. Once you have a list to store the area of each rectangles, you could use the function “sum()” to sum the total area and avoid the for loop.
4. Note that this method is memory inefficiency.

Performance Measurement

Timer and performance measurement

1. Performance measurement is very import in scientific computing.
2. Python could be VERY SLOW if we code it improperly.
3. It is very easy to make > 100 times slower than other programming language with python
4. When every you finish a program, you should think where is the performance bottleneck and whether we could improve it or not.

Timer and performance measurement

1. A simple way to measure the performance

```python
import time

def func1():
    # do some calculations
    a = 1
    for n in range(1000):
        a += 1
    print(a)
    return

if __name__=='__main__':

    t1 = time.time()
    func1()
    t2 = time.time()
    print(f"Time spend = {t2-t1}")
```

Timer and performance measurement

2. Use the “timeit” module

```
import timeit
timer = timeit.Timer(func1)
t = timer.timeit(number=10_000)
print(f"Average time = {t/10_000} sec.")
```

Timer and performance measurement

2. Use the “timeit” module to measure the performance of a function with arguments

```python
def func2(N=1_000):
    # do some calculations
    a = 1
    for n in range(N):
        a += 1
    return
```

```python
timer = timeit.Timer(lambda: func2(N=1000))
t = timer.timeit(number=1_000)
print(f"Average time = {t/1_000} sec.")
```

Performance measurement in Jupyter notebook

1. In Jupyter notebooks, use “%time” or “%timeit” to measure the performance.

```
%time a = func1(N=1_000_000)
```

✓ 0.0s

```
CPU times: user 42.5 ms, sys: 1.8 ms, total: 44.3 ms
Wall time: 42.8 ms
```

```
%timeit a = func1(N=1_000_000)
```

✓ 3.3s

```
40.6 ms ± 1.57 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Exercise: Create a list

1. Measure the performance of creating a list with N=1_000_000 ones (i.e. [1,1,1….,1]).
2. Use

```python
def make_list1(N=1_000_000):
    a = []
    for n in range(N):
        a.append(1)
```

3. Use

```python
def make_list2(N=1_000_000):
    a = [1 for n in range(N)]
    return a
```

4. Remember that the CPU clock speed is about ~GHz (or ~ 1ns per operation). We should expect that it takes only 1e6 (elements) x 1 ns ~ 1.e-6 sec to create the list. What are the values you observed? Is python fast?
5. (Demo): use numpy

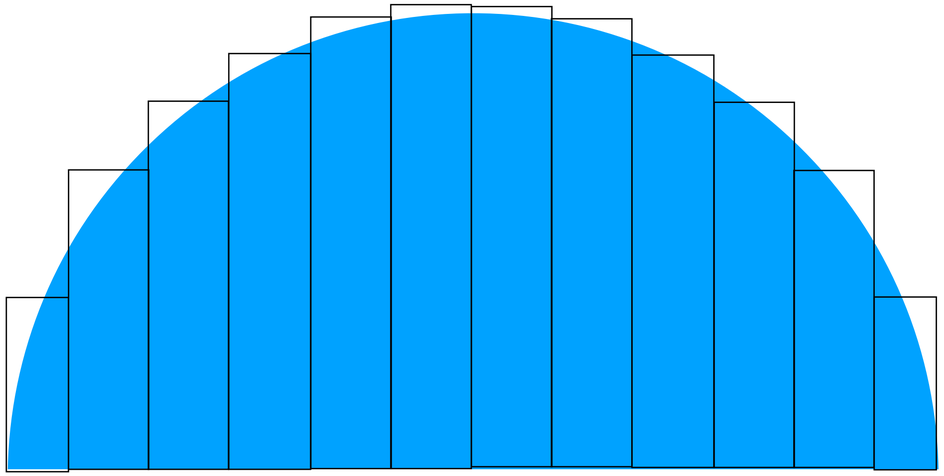
Exercise: Looping a list

1. Measure the performance of summing all the N=1_000_000 elements.
2. Use

```
def loop1(a):
    total = 0
    for v in a:
        total += v
    return total
```

3. Use the builtin “sum” function (i.e. total = sum(a))
4. (Demo): use numpy
5. Conclusions: Python’s default “loop” is VERY SLOW. In physical problem, we often encounter situation with big “array” or “matrix” (lists). We should avoid using loops. Scientific packages like “numpy” and “scipy” are therefore recommended.

Exercise: use lambda

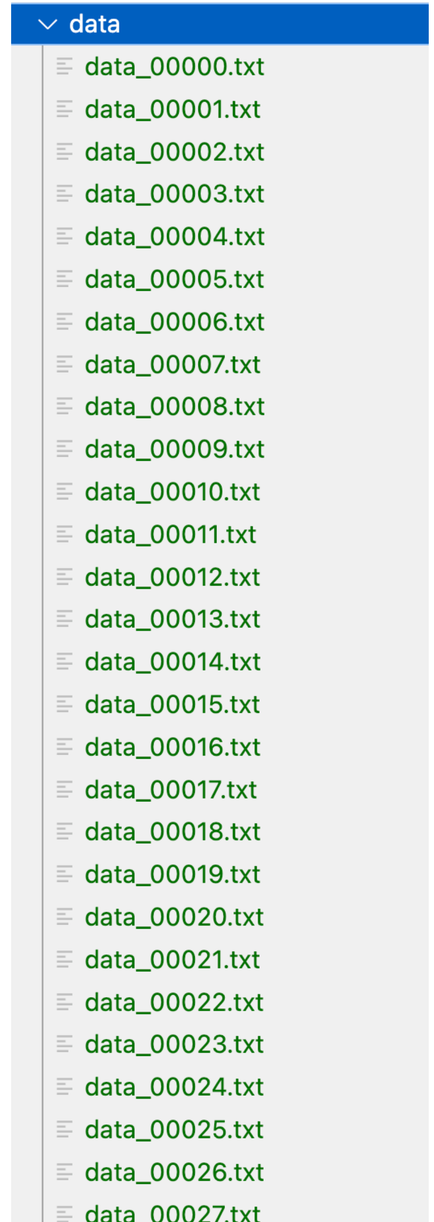


1. Back to your Integrator(), now your lambda to define your function
2. Write a new method called midpoint2. Use lambda or list comprehensions to avoid the for loop when summing the area
3. Measure the performance of the two methods (midpoint and midpoint2)
4. (we will show later that both methods are very slow)

File Wildcards: pathlib and glob

File wildcards: pathlib & glob

1. When doing data analysis, it is quite often that you need to handle multiple data files with sequent filenames (e.g. data_0001.txt, data_0002.txt, …etc).
2. str(intger).zfill(digits): convert a integer to a string and empty digits to zeros.
3. You might want to access all or part of the data files.
4. Search files that match a pattern string can be done by either “pathlib” or “glob”
5. pathlib: https://docs.python.org/3/library/pathlib.html
6. glob: https://docs.python.org/3/library/glob.html#module-glob



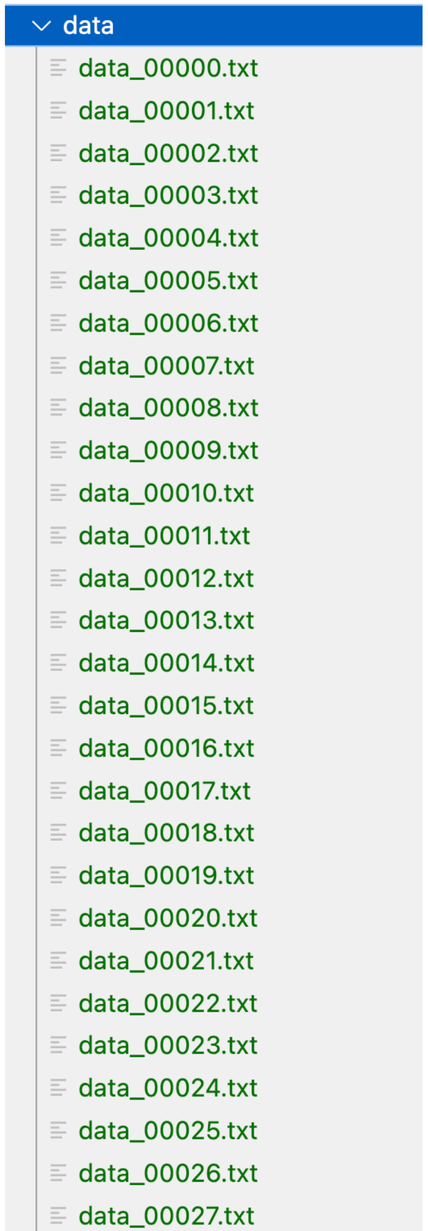
Example: pathlib & glob

1. Check if a folder exist. If not, create it.

```
from pathlib import Path

# check if the folder exists, if not create one
Path(folder).mkdir(parents=True, exist_ok=True)
```

2. Create sequent file names:

```
# loop all file names
for n in range(N):

    # the filename includes its relative path
    fname = folder+'/'+header+'_'+str(n).zfill(digits)+'.txt'
```



Example: pathlib & glob

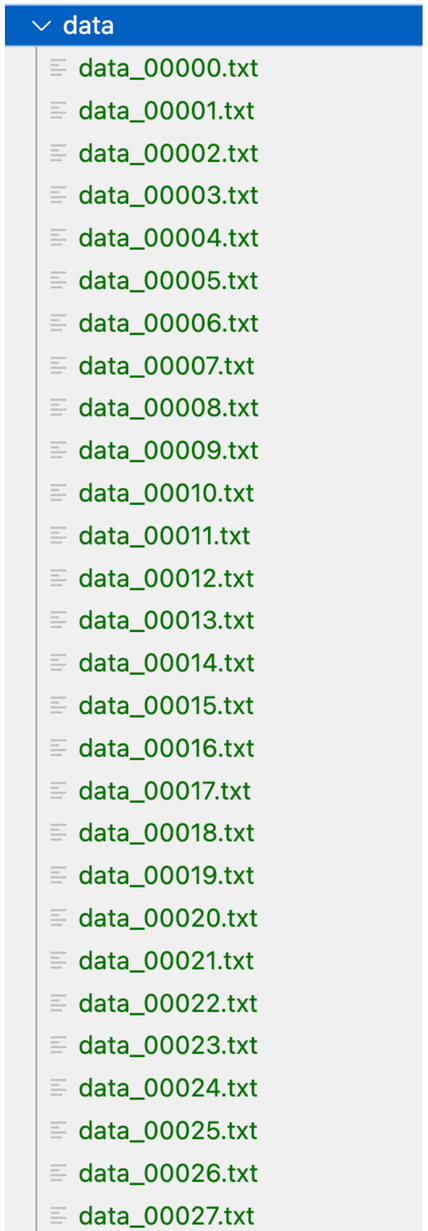
3. Get all file names that match a pattern

```
# get files that match the pattern
pattern = 'data_*[0,3,5][0-4].txt'
fns = sorted(Path(folder).glob(pattern))
print(fns)
```

4. Similar function can be achieved with glob.glob(pattern) as well.

```
pattern = folder+'/data_*[0,3,5][0-4].txt'
fns = sorted(glob.glob(pattern))
```

```
~/codes/ComputationalPhysics/ComputationalPhysics/tutorial (main*) » python tut_15_zfill.py                    pan@vega
[PosixPath('data/data_00000.txt'), PosixPath('data/data_00001.txt'), PosixPath('data/data_00002.txt'), PosixPath('data/data_00003.txt'), PosixPath('d
ata/data_00004.txt'), PosixPath('data/data_00030.txt'), PosixPath('data/data_00031.txt'), PosixPath('data/data_00032.txt'), PosixPath('data/data_0003
3.txt'), PosixPath('data/data_00034.txt'), PosixPath('data/data_00050.txt'), PosixPath('data/data_00051.txt'), PosixPath('data/data_00052.txt'), Posi
xPath('data/data_00053.txt'), PosixPath('data/data_00054.txt'), PosixPath('data/data_00100.txt'), PosixPath('data/data_00101.txt'), PosixPath('data/d
ata_00102.txt'), PosixPath('data/data_00103.txt'), PosixPath('data/data_00104.txt'), PosixPath('data/data_00130.txt'), PosixPath('data/data_00131.txt
'), PosixPath('data/data_00132.txt'), PosixPath('data/data_00133.txt'), PosixPath('data/data_00134.txt'), PosixPath('data/data_00150.txt'), PosixPath
('data/data_00151.txt'), PosixPath('data/data_00152.txt'), PosixPath('data/data_00153.txt'), PosixPath('data/data_00154.txt')]
```



Exercise: pathlib & glob

1. Run the sample code (tut_15_pathlib.py)

2. Experience the code with different N, header, and folder.

3. Play around with different “pattern”.

NumPy

NumPy

1. NumPy (Numerical Python) is an open source project that enables numerical computing with Python.
2. It was created in 2005 by Travis Oliphant.
3. Designed for good performance, core functionality is built around **arrays**, matrices, and linear algebra, …etc.
4. Installing NumPy: `conda install numpy`
5. Import NumPy: `import numpy as np`
6. Check NumPy version: `print(np.__version__)`

When to use numpy?

1. Dealing with arrays (mulit-D arrays)

2. Dealing with mathematical operations on arrays

3. Numerical calculations

NumPy array

1. Create a 1-d numpy array:

```
# create a numpy array
arr = np.arange(N)
```

2. Convert a python list to a numpy array

```
arr = np.array(list)
```

NumPy N-d array

3. Create a N-d numpy array:

```
def ndimensional_array():

    a0 = np.array(32)
    print(f"This is a {a0.ndim}-d numpy array: {a0}")
    a1 = np.array([0,1,2,3])
    print(f"This is a {a1.ndim}-d numpy array: {a1}")
    a2 = np.array([[1,2,3],[4,5,6]])
    print(f"This is a {a2.ndim}-d numpy array: {a2}")
    a3 = np.array([[[1,2,3],[4,5,6]],[[1,2,3],[4,5,6]]])
    print(f"This is a {a3.ndim}-d numpy array: {a3}")

    return
```

NumPy N-d array

4. Array shape:

```
print(a0.shape)
print(a1.shape)
print(a2.shape)
print(a3.shape)
```

```
()
(4,)
(2, 3)
(2, 2, 3)
```

NumPy N-d array

5. Array indexing:

```
print(a1[0])
print(a2[1,1])
print(a3[0,0,2])
```

6. Array slicing:

```
print(a1[0:2])
print(a2[1:2,1:2])
print(a3[0:1,0:1,0:2])
```

NumPy N-d array

7. Reshaping arrays:

```
arr1d = np.arange(12)
arr2d = arr1d.reshape(4,3)
arr3d = arr1d.reshape(2,2,3)

print(arr1d)
print(arr2d)
print(arr3d)
```

8. Flattening arrays:

```
arr1d  = arr3d.reshape(-1) # flatten
arr1d2 = arr3d.flatten()   # same as above
```

NumPy N-d array

9. Iterating N-d array

```
arr3d = np.arange(4*5*4).reshape(4,5,4)

# (method 1) iterate on the elements of a 3D array
for x in arr3d:
    for y in x:
        for z in y:
            print(z)

# (method 2) iterate on the elements of a 3D array
(nx,ny,nz) = arr3d.shape
print(nx,ny,nz)

for i in range(nx):
    for j in range(ny):
        for k in range(nz):
            print(arr3d[i,j,k])
```

```
# (method 3) we could also do it using nditer()
for v in np.nditer(arr3d):
    print(v)

# (method 4) nditer with index
for idx, x in np.ndenumerate(arr3d):
    print(idx, x)
```

Array creations

10. Creating arrays

```
a = np.zeros(shape=10)
print(a)
```

[2] ✓ 0.0s

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
b = np.ones(shape=(3,3))
print(b)
```

[3] ✓ 0.0s

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

```
c = np.linspace(0,1,10, endpoint=True)
print(c)
```

[5] ✓ 0.0s

```
[0.         0.11111111 0.22222222 0.33333333 0.44444444 0.55555556
 0.66666667 0.77777778 0.88888889 1.        ]
```

```
d = np.logspace(-1, 3, 10, endpoint=True)
print(d)
```

[10] ✓ 0.0s

```
[1.00000000e-01 2.78255940e-01 7.74263683e-01 2.15443469e+00
 5.99484250e+00 1.66810054e+01 4.64158883e+01 1.29154967e+02
 3.59381366e+02 1.00000000e+03]
```

NumPy: Array operation

11. Mathematical functions: https://numpy.org/doc/stable/reference/routines.math.html

12. Tips: If you are working on scalars, use the “math”, otherwise, use “numpy”.

13.

```
pylist  = [1,2,3,4,5,6,7,8,9,10]
nparray = np.array(pylist)

print(2*pylist)
print(2*nparray)
```

[42] ✓ 0.1s

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[ 2  4  6  8 10 12 14 16 18 20]
```

```
print(np.sin(nparray))
print(math.sin(pylist))
```

] ⊗ 0.0s

```
[ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427 -0.2794155
  0.6569866   0.98935825  0.41211849 -0.54402111]

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[46], line 2
      1 print(np.sin(nparray))
----> 2 print(math.sin(pylist))

TypeError: must be real number, not list
```

Exercise: numpy array

1. Numpy could do the numerical integration as well.
2. Check the documentation of numpy.trapz https://numpy.org/doc/stable/reference/generated/numpy.trapz.html
3. Use np.trapz() to do the pi calculation (see page 8).
4. Compare the results with your own numerical integrator.
5. Compare the performance with N=1_000_000.
6. The midpoint method is similar to the function np.sum()

NumPy: Random numbers

1. Random numbers are generated by random number generators.

2. Requires a “seed” to initialize the generator.

3.

```
[14]  import numpy.random as random
      ✓ 0.0s
```

```
[16]  random.seed(seed=123) # optional; used if you want to reproduce the results
      ✓ 0.0s
```

4. Random 10 integers from 0 to 100:

```
]     random.randint(0, 100, 10)
      ✓ 0.0s
```

```
array([52, 70, 26, 80,  6, 14, 75, 54, 71,  1])
```

NumPy: Random numbers

5. Random N float numbers from 0 to 1 (1d numpy array):

```
random.rand(10)
```

✓ 0.0s

```
array([0.48303426, 0.98555979, 0.51948512, 0.61289453, 0.12062867,
       0.8263408 , 0.60306013, 0.54506801, 0.34276383, 0.30412079])
```

6. Random 5x5 float numbers from 0 to 1 (2d numpy array):

```
random.rand(5,5)
```

✓ 0.0s

```
array([[0.41702221, 0.68130077, 0.87545684, 0.51042234, 0.66931378],
       [0.58593655, 0.6249035 , 0.67468905, 0.84234244, 0.08319499],
       [0.76368284, 0.24366637, 0.19422296, 0.57245696, 0.09571252],
       [0.88532683, 0.62724897, 0.72341636, 0.01612921, 0.59443188],
       [0.55678519, 0.15895964, 0.15307052, 0.69552953, 0.31876643]])
```

NumPy: Random numbers

7. Generate a 1-D array containing 10 values, where each value has to be 1,2,3,4,5.

Probability distribution is given to be p = [0.2,0.2,0.3,0.15,0.15]

```
random.choice([1,2,3,4,5], p = [0.2, 0.2, 0.3, 0.15, 0.15], size= 10)
```

✓ 0.0s

```
array([4, 1, 1, 2, 5, 2, 3, 3, 2, 3])
```

6. Shuffle a numpy array:

```
students = ["Einstein", "Newton", "Maxwell"]
random.shuffle(students)
print(students)
```

✓ 0.0s

```
['Maxwell', 'Newton', 'Einstein']
```

Exercise: Random numbers

1. Write a python class named “Gacha” (轉蛋).

2. The class has three attributes, which are numpy arrays. Each array has five elements:

```
self.stars3 = np.array(["a1","a2","a3","a4","a5"])
self.stars4 = np.array(["b1","b2", "b3","b4","b5"])
self.stars5 = np.array(["s1","s2","s3","s4","s5"])
```

3. Write a method called “draw()”. It will return a tuple (star, char), which “star” is an integer from 3 to 5; “char” is a string randomly taken from star3/star4/star5 if star=3/4/5.

4. The probability to receive star=5, star=4, and star=3 are 1%, 9%, and 90%, respectively.

5. Draw 100 times, measure how many 5 stars did you get and how many “s1”?