PYTHON - <u>File IO using Python :</u>

Printing to the Screen

The simplest way to produce output is using the print statement where you can pass zero or more expressions separated by commas. This function converts the expressions you pass into a string and writes the result to standard output as follows −

```
print "Python is really a great language,", "isn't it?"
```

This produces the following result on your standard screen −

```
Python is really a great language, isn't it?
```

Reading Keyboard Input

Python provides two built-in functions to read a line of text from standard input, which by default comes from the keyboard. These functions are −

raw_input
input

1. The raw_input Function

The raw_input([prompt]) function reads one line from standard input and returns it as a string

```
str = raw_input("Enter your input: ")
print "Received input is : ", str
```

This prompts you to enter any string and it would display same string on the screen. When I typed "Hello Python!", its output is like this –

```
Enter your input: Hello Python
Received input is :  Hello Python
```

- Enriching Data using Numpy & Pandas

What is NumPy?

➢ NumPy = Numerical Python

It’s a Python library used to do fast math with arrays (like lists but more powerful).

```
# Converting a Python list to a NumPy array:

import numpy as np

cvalues = [20.1, 20.8, 20.5, 22.8, 24.5, 22.9]

data = np.array(cvalues)

print(data)

#This is now a NumPy array, not a list.

Output :

PS C:\Users\masih\OneDrive\py coding> python data_num.py

[20.1 20.8 20.5 22.8 24.5 22.9]

#Example: Celsius to Fahrenheit

cvalues = [20.1, 20.8, 21.9, 22.5, 22.7, 22.3, 21.8, 21.2, 20.9, 20.1]

C = np.array(cvalues)
```

```
print(C * 9 / 5 + 32)

#That’s it. That line converts every element in the array from Celsius to
Fahrenheit.

Output :

PS C:\Users\masih\OneDrive\py coding> python data_num.py

[68.18 69.44 71.42 72.5  72.86 72.14 71.24 70.16 69.62 68.18]
```

Compare it to the old-school Python list way:

fvalues = [x * 9 / 5 + 32 for x in cvalues]

NumPy saves you from loops. It's cleaner and faster.

type(C)

You use type() in Python to check what kind of object something is.

So when you write:

```
import numpy as np

cvalues = [20.1, 20.8, 21.9]

C = np.array(cvalues)

print(type(C))

Output: PS C:\Users\masih\OneDrive\py coding> python data_num.py

<class 'numpy.ndarray'>
```

What does `<class 'numpy.ndarray'>` mean?

Let's break it down:

- **class**: This means C is an object **created from a class** (a blueprint in Python).
- **'numpy.ndarray'**:
    - **numpy**: The library that created this object.
    - **nd array**: This is short for **n-dimensional array**.

What is an "n-dimensional array"?

An array is just a collection of numbers — like a list.

- **1D array** = a simple line of numbers
  Example: `[1, 2, 3]`
- **2D array** = like a table or grid (rows & columns)
  Example:
  ```
  [[1, 2, 3],
  [4, 5, 6]]
  ```
- **3D array** = multiple 2D arrays stacked together (like pages in a book)

The **"n"** in **n-dimensional** means it can be **any number of dimensions** — 1D, 2D, 3D, or more.

NumPy: Creating Arrays with Evenly Spaced Values

There are **two main functions** you'll use for this:

1. `np.arange()` ➜ You choose the **step size**
2. `np.linspace()` ➜ You choose the **number of values**

One 'arange' uses a given distance and the other one 'linspace' needs the number of elements and creates the distance automatically.

1. `np.arange()` — Like `range()` but smarter

📌 What does it do?

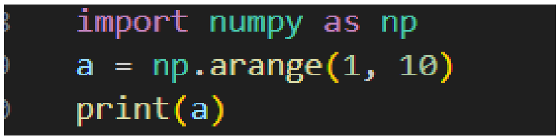
It creates a NumPy array by counting in steps from a starting number to a stopping number (but not including the stop number).

Syntax:

`np.arange(start, stop, step, dtype)`

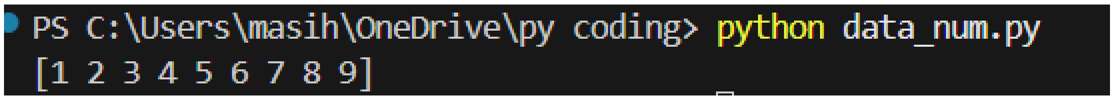
- `start`: (optional) where to begin. The default is 0.
- `stop`: (required) where to stop (but stop itself is **not included**).
- `step`: (optional) how much to add each time (default is 1).
- `dtype`: (optional) type of data (int, float, etc.)

Examples:



```python
import numpy as np
a = np.arange(1, 10)
print(a)
```

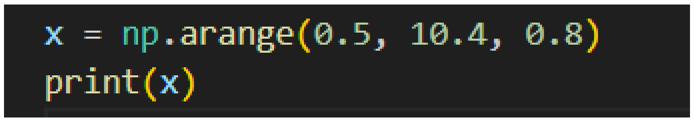
Output :



```
PS C:\Users\masih\OneDrive\py coding> python data_num.py
[1 2 3 4 5 6 7 8 9]
```

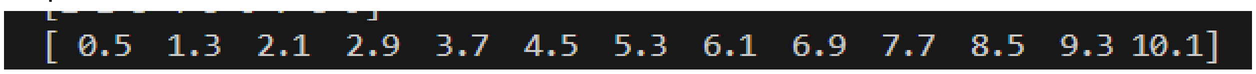
This counts from 1 to 9 (but not 10), step is 1.

Float values



```python
x = np.arange(0.5, 10.4, 0.8)
print(x)
```

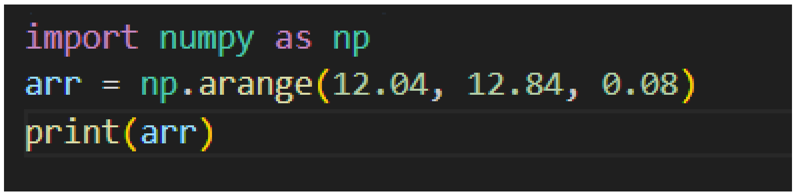
Output:



```
[ 0.5  1.3  2.1  2.9  3.7  4.5  5.3  6.1  6.9  7.7  8.5  9.3 10.1]
```

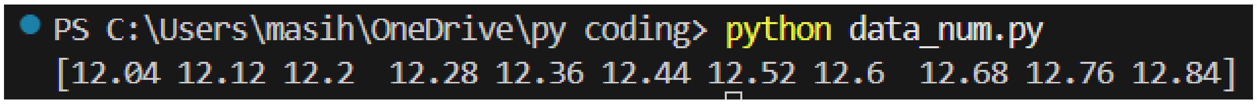
Each value increases by 0.8.
Warning: When using **decimal steps**, results can look weird because of floating point rounding.

Example of rounding issues



```python
import numpy as np
arr = np.arange(12.04, 12.84, 0.08)
print(arr)
```

Output:



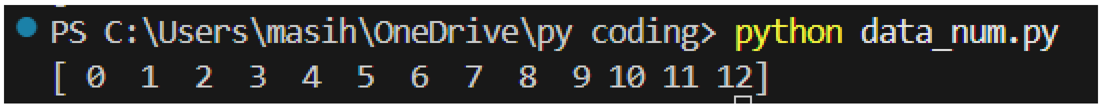
```
PS C:\Users\masih\OneDrive\py coding> python data_num.py
[12.04 12.12 12.2  12.28 12.36 12.44 12.52 12.6  12.68 12.76 12.84]
```

Even though `12.84` is the stop, it's **included** here — that's **not normal** — it's because of how floating point math works inside computers.

Weird case: forcing output to be integers

```
x = np.arange(0.5, 10.4, 0.8, int)
print(x)
```

Output:



```
PS C:\Users\masih\OneDrive\py coding> python data_num.py
[ 0  1  2  3  4  5  6  7  8  9 10 11 12]
```

Why is it so weird?

- It starts at `0.5` but rounds down to `0`
- It keeps going with steps of 0.8, but **each value gets converted to an integer**
- So you get `[0, 1, 2, 3, ..., 12]`

This is why the docs say:

"When using non-integer step like 0.1, results can be unpredictable. Use `linspace()` instead."

2. `np.linspace()` — Choose number of points (not step)

📌 What does it do?

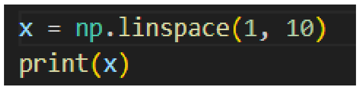
It creates a NumPy array of **evenly spaced values** from a start number to a stop number.

Syntax:

`np.linspace(start, stop, num=50, endpoint=True, retstep=False)`

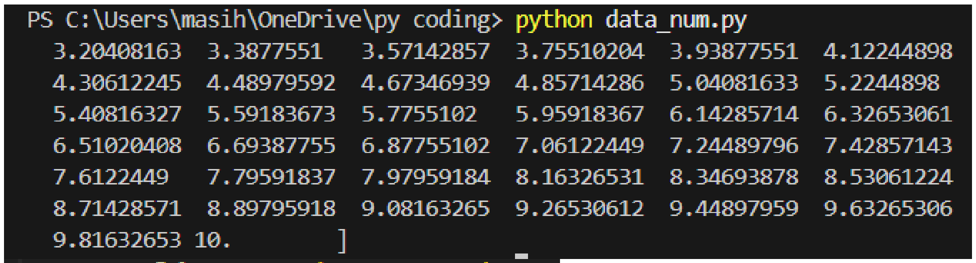
- `start`: where to begin
- `stop`: where to end
- num: how many values to generate
- `endpoint`: if `True`, include the stop value
- `retstep`: if `True`, return spacing between values too

Examples:

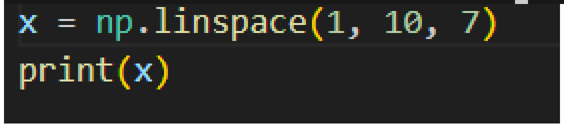


```
x = np.linspace(1, 10)
print(x)
```

Output :

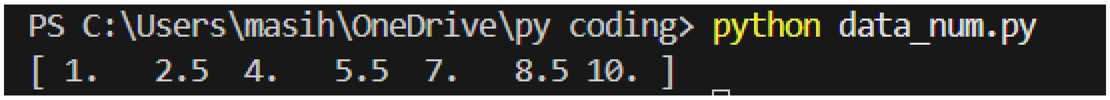


```
PS C:\Users\masih\OneDrive\py coding> python data_num.py
  3.20408163  3.3877551   3.57142857  3.75510204  3.93877551  4.12244898
  4.30612245  4.48979592  4.67346939  4.85714286  5.04081633  5.2244898
  5.40816327  5.59183673  5.7755102   5.95918367  6.14285714  6.32653061
  6.51020408  6.69387755  6.87755102  7.06122449  7.24489796  7.42857143
  7.6122449   7.79591837  7.97959184  8.16326531  8.34693878  8.53061224
  8.71428571  8.89795918  9.08163265  9.26530612  9.44897959  9.63265306
  9.81632653 10.        ]
```



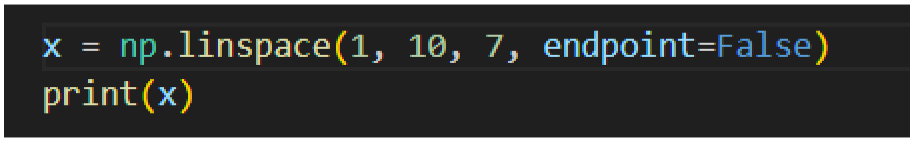
```
x = np.linspace(1, 10, 7)
print(x)
```

Output :



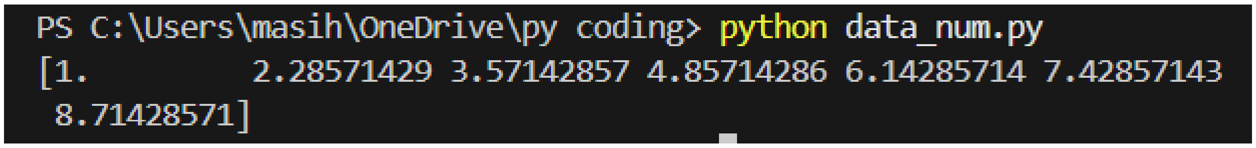
```
PS C:\Users\masih\OneDrive\py coding> python data_num.py
[ 1.   2.5  4.   5.5  7.   8.5 10. ]
```

7 values equally spaced from 1 to 10.



```
x = np.linspace(1, 10, 7, endpoint=False)
print(x)
```

Output :

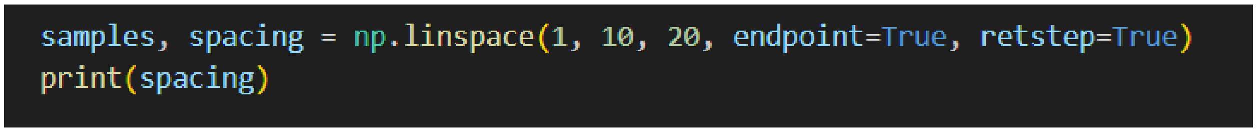


```
PS C:\Users\masih\OneDrive\py coding> python data_num.py
[1.         2.28571429 3.57142857 4.85714286 6.14285714 7.42857143
 8.71428571]
```

Here, 10 is not included because `endpoint=False`.

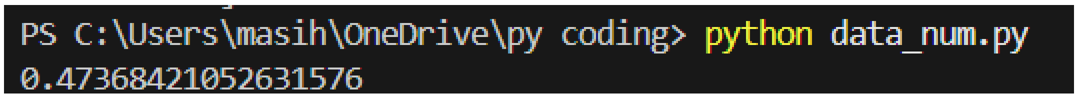
Want to know the gap between numbers?

Use `retstep=True`:



```python
samples, spacing = np.linspace(1, 10, 20, endpoint=True, retstep=True)
print(spacing)
```

Output:



```
PS C:\Users\masih\OneDrive\py coding> python data_num.py
0.47368421052631576
```

This means each number increases by `~0.47`.

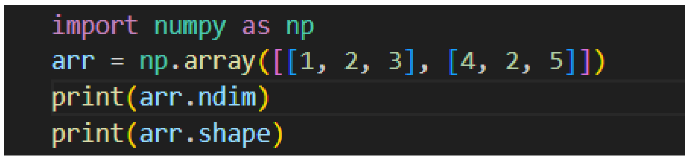
Feature	`np.arange()`	`np.linspace()`
You give	**start, stop, step**	**start, stop, number of values**
Step size	**You choose manually**	**Calculated automatically**
Includes stop?	❌ **Usually not**	✅ **If `endpoint=True` (default)**
Good for	**Integer steps (1, 2, 3, …)**	**Decimal values without weird rounding**

Returns	**NumPy array**	**NumPy array (can return spacing too)**

Arrays in NumPy:

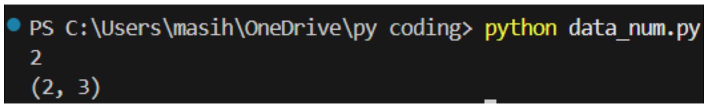
- The main object is `ndarray` (N-dimensional array).
- It stores data in a **grid** (like tables) where all elements are the **same type**.
- "Axis" = dimension. For example, a 2D array has 2 axes (rows and columns).

Example:



```python
import numpy as np
arr = np.array([[1, 2, 3], [4, 2, 5]])
print(arr.ndim)
print(arr.shape)
```

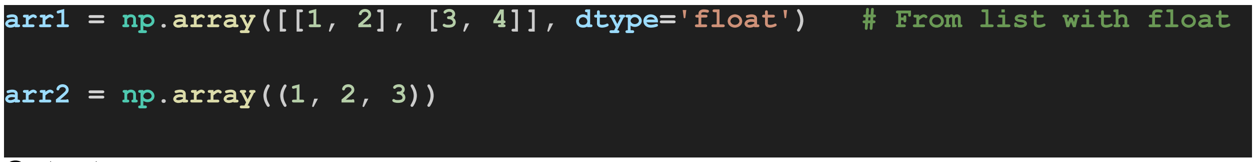
Output :



```
PS C:\Users\masih\OneDrive\py coding> python data_num.py
2
(2, 3)
```

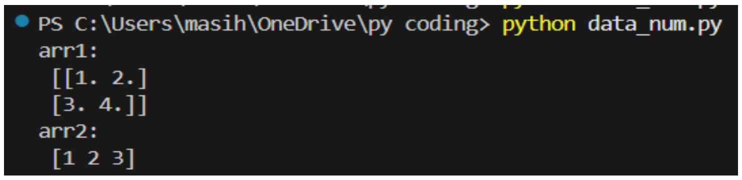
Array Creation Methods:

1. **From Lists or Tuples**



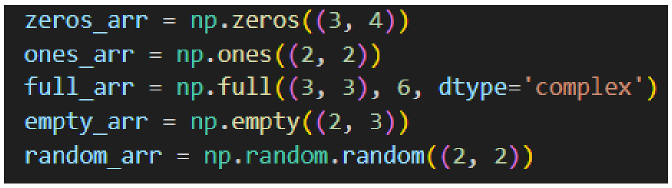
```
arr1 = np.array([[1, 2], [3, 4]], dtype='float')   # From list with float

arr2 = np.array((1, 2, 3))
```

Output :



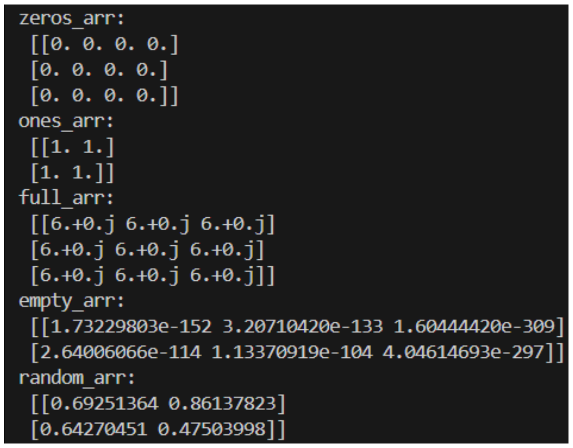
```
PS C:\Users\masih\OneDrive\py coding> python data_num.py
arr1:
 [[1. 2.]
 [3. 4.]]
arr2:
 [1 2 3]
```

2. **Using Placeholder Arrays**



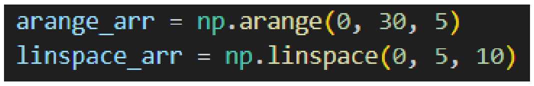
```
zeros_arr = np.zeros((3, 4))
ones_arr = np.ones((2, 2))
full_arr = np.full((3, 3), 6, dtype='complex')
empty_arr = np.empty((2, 3))
random_arr = np.random.random((2, 2))
```

Output :



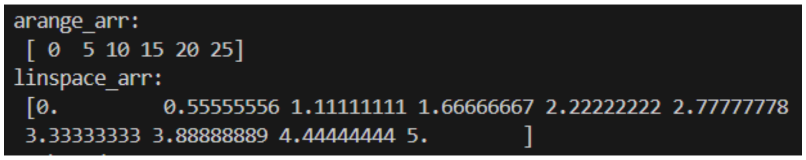
```
zeros_arr:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
ones_arr:
 [[1. 1.]
 [1. 1.]]
full_arr:
 [[6.+0.j 6.+0.j 6.+0.j]
 [6.+0.j 6.+0.j 6.+0.j]
 [6.+0.j 6.+0.j 6.+0.j]]
empty_arr:
 [[1.73229803e-152 3.20710420e-133 1.60444420e-309]
 [2.64006066e-114 1.13370919e-104 4.04614693e-297]]
random_arr:
 [[0.69251364 0.86137823]
 [0.64270451 0.47503998]]
```

3. **Using Sequences**



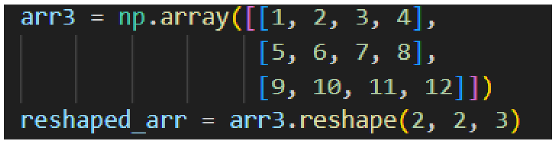
```
arange_arr = np.arange(0, 30, 5)
linspace_arr = np.linspace(0, 5, 10)
```

Output :



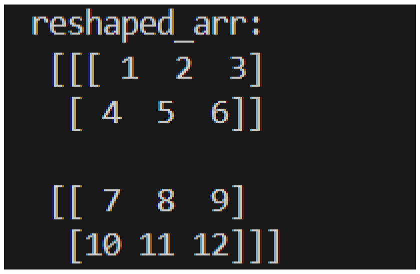
```
arange_arr:
 [ 0  5 10 15 20 25]
linspace_arr:
 [0.         0.55555556 1.11111111 1.66666667 2.22222222 2.77777778
 3.33333333 3.88888889 4.44444444 5.        ]
```

4. **Reshaping Arrays**



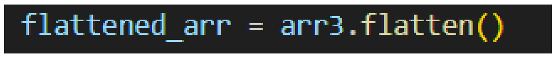
```python
arr3 = np.array([[1, 2, 3, 4],
                 [5, 6, 7, 8],
                 [9, 10, 11, 12]])
reshaped_arr = arr3.reshape(2, 2, 3)
```

Output :



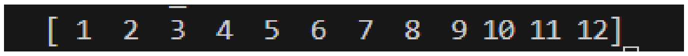
```
reshaped_arr:
 [[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]
```

5. **Flatten Array (1D view)**



```python
flattened_arr = arr3.flatten()
```

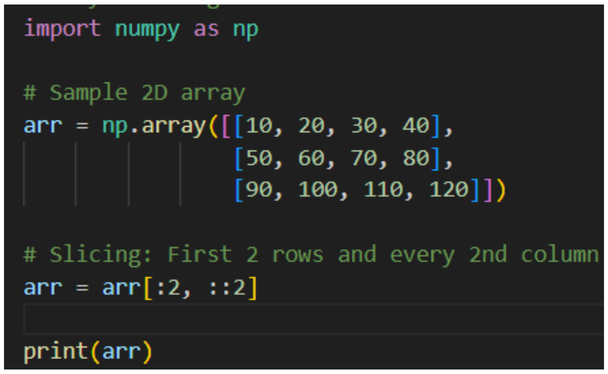
Output :



```
 [ 1  2  3  4  5  6  7  8  9 10 11 12]
```

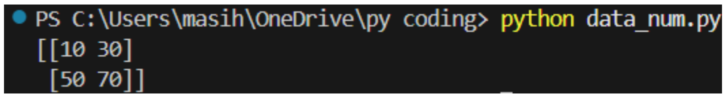
Array Indexing:

Slicing



```python
import numpy as np

# Sample 2D array
arr = np.array([[10, 20, 30, 40],
                [50, 60, 70, 80],
                [90, 100, 110, 120]])

# Slicing: First 2 rows and every 2nd column
arr = arr[:2, ::2]

print(arr)
```

Output :



```
PS C:\Users\masih\OneDrive\py coding> python data_num.py
[[10 30]
 [50 70]]
```