1-intro-numpy

October 22, 2020

1 Introducción

NumPy es la *librería* de python para computación científica. **NumPy** agrega al lenguaje lo siguiente: Arreglos multidimensionales, operaciones elemento por elemento (técnica conocida como *broadcasting*), algebra lineal, manipulación de imágenes, la habilidad de utilizar código `C/C++` y `FORTRAN`, entre muchas otras.

La mayor parte de los componentes del sistema de computo científico de Python, están construidas encima de **NumPy**, un ejemplo que veremos en el curso es `SciPy`.

Para poder utilizar **NumPy**, es necesario importarlo a la sesión del `notebook`.

1.0.1 Bibliografía de soporte

- *NumPy Beginner's Guide Ivan Idris*, PACKT Publishing, 2012
- *NumPy Cookbook Ivan Idris*, PACKT Publishing, 2012
- *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython, Wes McKinney*, O'REILLY, 2012

[1]:
```
import numpy as np
```

1.1 Arrays

El principal componente de **NumPy** es el `array`, el cual es una versión más poderosa, pero menos flexible que las listas de python.

[2]:
```
lst =  [1,2,3,4,5]
lst
```

[2]:
```
[1, 2, 3, 4, 5]
```

[3]:
```
arr = np.array([1,2,3,4,5])
arr
```

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

[4]:
```
lst[1:3]
```

[4]:
```
[2, 3]
```

[5]:
```
arr[1:3]
```

[5]:
```
array([2, 3])
```

Ejercicio: Repite los ejercicios de listas del *Lecture 2* con `array`.

[6]:
```
lst[-1] = "Las listas pueden tener varios tipos de datos"
lst
```

[6]:
```
[1, 2, 3, 4, 'Las listas pueden tener varios tipos de datos']
```

[7]:
```
arr[-1] = "Los arreglos no..."
#No se pueden poner varios tipos de datos
```

```
␣
↪---------------------------------------------------------------------------

       ValueError                                Traceback (most recent call␣
↪last)

       <ipython-input-7-267523e227b5> in <module>
   ----> 1 arr[-1] = "Los arreglos no…"
         2 #No se pueden poner varios tipos de datos


       ValueError: invalid literal for int() with base 10: 'Los arreglos no…'
```

Una vez inicializado el `array` sólo puede contener un tipo de dato.

[8]:
```
arr.dtype
#Es un entero de 64 bytes
```

[8]:
```
dtype('int64')
```

[9]:
```
arr[-1] = 1.23456
arr
```

[9]:
```
array([1, 2, 3, 4, 1])
```

[10]:
```
arr.dtype
```

[10]:
```
dtype('int64')
```

Sacrificamos la versatilidad de las listas por velocidad. Creeemos un `array` de 1 millón de elementos y multiplicaremos cada uno de ellos por una constante (*broadcasting*). –> hacerle operaciones a todos los elementos

[13]:
```
arr = np.arange(1e7)
arr
```

[13]:
```
array([0.000000e+00, 1.000000e+00, 2.000000e+00, …, 9.999997e+06,
       9.999998e+06, 9.999999e+06])
```

[15]:
```
#Convierte un arreglo a lista
lst = arr.tolist()
lst
```

[15]:
```
[0.0,
 1.0,
 2.0,
 3.0,
 4.0,
 5.0,
 6.0,
 7.0,
 8.0,
 9.0,
 10.0,
 11.0,
 12.0,
 13.0,
 14.0,
 15.0,
 16.0,
 17.0,
 18.0,
 19.0,
 20.0,
 21.0,
 22.0,
 23.0,
 24.0,
 25.0,
 26.0,
 27.0,
 28.0,
 29.0,
 30.0,
 31.0,
 32.0,
 33.0,
 34.0,
 35.0,
 36.0,
```

```
37.0,
38.0,
39.0,
40.0,
41.0,
42.0,
43.0,
44.0,
45.0,
46.0,
47.0,
48.0,
49.0,
50.0,
51.0,
52.0,
53.0,
54.0,
55.0,
56.0,
57.0,
58.0,
59.0,
60.0,
61.0,
62.0,
63.0,
64.0,
65.0,
66.0,
67.0,
68.0,
69.0,
70.0,
71.0,
72.0,
73.0,
74.0,
75.0,
76.0,
77.0,
78.0,
79.0,
80.0,
81.0,
82.0,
83.0,
```

84.0,
85.0,
86.0,
87.0,
88.0,
89.0,
90.0,
91.0,
92.0,
93.0,
94.0,
95.0,
96.0,
97.0,
98.0,
99.0,
100.0,
101.0,
102.0,
103.0,
104.0,
105.0,
106.0,
107.0,
108.0,
109.0,
110.0,
111.0,
112.0,
113.0,
114.0,
115.0,
116.0,
117.0,
118.0,
119.0,
120.0,
121.0,
122.0,
123.0,
124.0,
125.0,
126.0,
127.0,
128.0,
129.0,
130.0,

131.0,
132.0,
133.0,
134.0,
135.0,
136.0,
137.0,
138.0,
139.0,
140.0,
141.0,
142.0,
143.0,
144.0,
145.0,
146.0,
147.0,
148.0,
149.0,
150.0,
151.0,
152.0,
153.0,
154.0,
155.0,
156.0,
157.0,
158.0,
159.0,
160.0,
161.0,
162.0,
163.0,
164.0,
165.0,
166.0,
167.0,
168.0,
169.0,
170.0,
171.0,
172.0,
173.0,
174.0,
175.0,
176.0,
177.0,

178.0,
179.0,
180.0,
181.0,
182.0,
183.0,
184.0,
185.0,
186.0,
187.0,
188.0,
189.0,
190.0,
191.0,
192.0,
193.0,
194.0,
195.0,
196.0,
197.0,
198.0,
199.0,
200.0,
201.0,
202.0,
203.0,
204.0,
205.0,
206.0,
207.0,
208.0,
209.0,
210.0,
211.0,
212.0,
213.0,
214.0,
215.0,
216.0,
217.0,
218.0,
219.0,
220.0,
221.0,
222.0,
223.0,
224.0,

225.0,
226.0,
227.0,
228.0,
229.0,
230.0,
231.0,
232.0,
233.0,
234.0,
235.0,
236.0,
237.0,
238.0,
239.0,
240.0,
241.0,
242.0,
243.0,
244.0,
245.0,
246.0,
247.0,
248.0,
249.0,
250.0,
251.0,
252.0,
253.0,
254.0,
255.0,
256.0,
257.0,
258.0,
259.0,
260.0,
261.0,
262.0,
263.0,
264.0,
265.0,
266.0,
267.0,
268.0,
269.0,
270.0,
271.0,

272.0,
273.0,
274.0,
275.0,
276.0,
277.0,
278.0,
279.0,
280.0,
281.0,
282.0,
283.0,
284.0,
285.0,
286.0,
287.0,
288.0,
289.0,
290.0,
291.0,
292.0,
293.0,
294.0,
295.0,
296.0,
297.0,
298.0,
299.0,
300.0,
301.0,
302.0,
303.0,
304.0,
305.0,
306.0,
307.0,
308.0,
309.0,
310.0,
311.0,
312.0,
313.0,
314.0,
315.0,
316.0,
317.0,
318.0,

```
319.0,
320.0,
321.0,
322.0,
323.0,
324.0,
325.0,
326.0,
327.0,
328.0,
329.0,
330.0,
331.0,
332.0,
333.0,
334.0,
335.0,
336.0,
337.0,
338.0,
339.0,
340.0,
341.0,
342.0,
343.0,
344.0,
345.0,
346.0,
347.0,
348.0,
349.0,
350.0,
351.0,
352.0,
353.0,
354.0,
355.0,
356.0,
357.0,
358.0,
359.0,
360.0,
361.0,
362.0,
363.0,
364.0,
365.0,
```

366.0,
367.0,
368.0,
369.0,
370.0,
371.0,
372.0,
373.0,
374.0,
375.0,
376.0,
377.0,
378.0,
379.0,
380.0,
381.0,
382.0,
383.0,
384.0,
385.0,
386.0,
387.0,
388.0,
389.0,
390.0,
391.0,
392.0,
393.0,
394.0,
395.0,
396.0,
397.0,
398.0,
399.0,
400.0,
401.0,
402.0,
403.0,
404.0,
405.0,
406.0,
407.0,
408.0,
409.0,
410.0,
411.0,
412.0,

413.0,
414.0,
415.0,
416.0,
417.0,
418.0,
419.0,
420.0,
421.0,
422.0,
423.0,
424.0,
425.0,
426.0,
427.0,
428.0,
429.0,
430.0,
431.0,
432.0,
433.0,
434.0,
435.0,
436.0,
437.0,
438.0,
439.0,
440.0,
441.0,
442.0,
443.0,
444.0,
445.0,
446.0,
447.0,
448.0,
449.0,
450.0,
451.0,
452.0,
453.0,
454.0,
455.0,
456.0,
457.0,
458.0,
459.0,

460.0,
461.0,
462.0,
463.0,
464.0,
465.0,
466.0,
467.0,
468.0,
469.0,
470.0,
471.0,
472.0,
473.0,
474.0,
475.0,
476.0,
477.0,
478.0,
479.0,
480.0,
481.0,
482.0,
483.0,
484.0,
485.0,
486.0,
487.0,
488.0,
489.0,
490.0,
491.0,
492.0,
493.0,
494.0,
495.0,
496.0,
497.0,
498.0,
499.0,
500.0,
501.0,
502.0,
503.0,
504.0,
505.0,
506.0,

```
507.0,
508.0,
509.0,
510.0,
511.0,
512.0,
513.0,
514.0,
515.0,
516.0,
517.0,
518.0,
519.0,
520.0,
521.0,
522.0,
523.0,
524.0,
525.0,
526.0,
527.0,
528.0,
529.0,
530.0,
531.0,
532.0,
533.0,
534.0,
535.0,
536.0,
537.0,
538.0,
539.0,
540.0,
541.0,
542.0,
543.0,
544.0,
545.0,
546.0,
547.0,
548.0,
549.0,
550.0,
551.0,
552.0,
553.0,
```

554.0,
555.0,
556.0,
557.0,
558.0,
559.0,
560.0,
561.0,
562.0,
563.0,
564.0,
565.0,
566.0,
567.0,
568.0,
569.0,
570.0,
571.0,
572.0,
573.0,
574.0,
575.0,
576.0,
577.0,
578.0,
579.0,
580.0,
581.0,
582.0,
583.0,
584.0,
585.0,
586.0,
587.0,
588.0,
589.0,
590.0,
591.0,
592.0,
593.0,
594.0,
595.0,
596.0,
597.0,
598.0,
599.0,
600.0,

601.0,
602.0,
603.0,
604.0,
605.0,
606.0,
607.0,
608.0,
609.0,
610.0,
611.0,
612.0,
613.0,
614.0,
615.0,
616.0,
617.0,
618.0,
619.0,
620.0,
621.0,
622.0,
623.0,
624.0,
625.0,
626.0,
627.0,
628.0,
629.0,
630.0,
631.0,
632.0,
633.0,
634.0,
635.0,
636.0,
637.0,
638.0,
639.0,
640.0,
641.0,
642.0,
643.0,
644.0,
645.0,
646.0,
647.0,

```
648.0,
649.0,
650.0,
651.0,
652.0,
653.0,
654.0,
655.0,
656.0,
657.0,
658.0,
659.0,
660.0,
661.0,
662.0,
663.0,
664.0,
665.0,
666.0,
667.0,
668.0,
669.0,
670.0,
671.0,
672.0,
673.0,
674.0,
675.0,
676.0,
677.0,
678.0,
679.0,
680.0,
681.0,
682.0,
683.0,
684.0,
685.0,
686.0,
687.0,
688.0,
689.0,
690.0,
691.0,
692.0,
693.0,
694.0,
```

695.0,
696.0,
697.0,
698.0,
699.0,
700.0,
701.0,
702.0,
703.0,
704.0,
705.0,
706.0,
707.0,
708.0,
709.0,
710.0,
711.0,
712.0,
713.0,
714.0,
715.0,
716.0,
717.0,
718.0,
719.0,
720.0,
721.0,
722.0,
723.0,
724.0,
725.0,
726.0,
727.0,
728.0,
729.0,
730.0,
731.0,
732.0,
733.0,
734.0,
735.0,
736.0,
737.0,
738.0,
739.0,
740.0,
741.0,

742.0,
743.0,
744.0,
745.0,
746.0,
747.0,
748.0,
749.0,
750.0,
751.0,
752.0,
753.0,
754.0,
755.0,
756.0,
757.0,
758.0,
759.0,
760.0,
761.0,
762.0,
763.0,
764.0,
765.0,
766.0,
767.0,
768.0,
769.0,
770.0,
771.0,
772.0,
773.0,
774.0,
775.0,
776.0,
777.0,
778.0,
779.0,
780.0,
781.0,
782.0,
783.0,
784.0,
785.0,
786.0,
787.0,
788.0,

```
789.0,
790.0,
791.0,
792.0,
793.0,
794.0,
795.0,
796.0,
797.0,
798.0,
799.0,
800.0,
801.0,
802.0,
803.0,
804.0,
805.0,
806.0,
807.0,
808.0,
809.0,
810.0,
811.0,
812.0,
813.0,
814.0,
815.0,
816.0,
817.0,
818.0,
819.0,
820.0,
821.0,
822.0,
823.0,
824.0,
825.0,
826.0,
827.0,
828.0,
829.0,
830.0,
831.0,
832.0,
833.0,
834.0,
835.0,
```

836.0,
837.0,
838.0,
839.0,
840.0,
841.0,
842.0,
843.0,
844.0,
845.0,
846.0,
847.0,
848.0,
849.0,
850.0,
851.0,
852.0,
853.0,
854.0,
855.0,
856.0,
857.0,
858.0,
859.0,
860.0,
861.0,
862.0,
863.0,
864.0,
865.0,
866.0,
867.0,
868.0,
869.0,
870.0,
871.0,
872.0,
873.0,
874.0,
875.0,
876.0,
877.0,
878.0,
879.0,
880.0,
881.0,
882.0,

883.0,
884.0,
885.0,
886.0,
887.0,
888.0,
889.0,
890.0,
891.0,
892.0,
893.0,
894.0,
895.0,
896.0,
897.0,
898.0,
899.0,
900.0,
901.0,
902.0,
903.0,
904.0,
905.0,
906.0,
907.0,
908.0,
909.0,
910.0,
911.0,
912.0,
913.0,
914.0,
915.0,
916.0,
917.0,
918.0,
919.0,
920.0,
921.0,
922.0,
923.0,
924.0,
925.0,
926.0,
927.0,
928.0,
929.0,

```
930.0,
931.0,
932.0,
933.0,
934.0,
935.0,
936.0,
937.0,
938.0,
939.0,
940.0,
941.0,
942.0,
943.0,
944.0,
945.0,
946.0,
947.0,
948.0,
949.0,
950.0,
951.0,
952.0,
953.0,
954.0,
955.0,
956.0,
957.0,
958.0,
959.0,
960.0,
961.0,
962.0,
963.0,
964.0,
965.0,
966.0,
967.0,
968.0,
969.0,
970.0,
971.0,
972.0,
973.0,
974.0,
975.0,
976.0,
```

```
977.0,
978.0,
979.0,
980.0,
981.0,
982.0,
983.0,
984.0,
985.0,
986.0,
987.0,
988.0,
989.0,
990.0,
991.0,
992.0,
993.0,
994.0,
995.0,
996.0,
997.0,
998.0,
999.0,
…]
```

Las listas no soportan **broadcasting** por lo que crearemos una función que lo simule

[16]:
```
def lst_multiplicacion( alist , scalar ):
    for i , val in enumerate ( alist ):
        alist [ i ] = val
    return alist
```

[17]:
```
%timeit arr * 1.1
```

```
19.3 ms ± 2.72 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

[18]:
```
%timeit lst_multiplicacion(lst, 1.1)
```

```
1.38 s ± 257 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

1.2 Creación de arrays

[19]:
```
arr = np.array([1,2,3,4,5]) #Convierte lista en arreglo
```

[20]:
```
arr
```

[20]:
```
array([1, 2, 3, 4, 5])
```

[21]:
```
arr = np.arange(10,21) #Convierte range en arregle
```

[22]:
```
arr
```

[22]:
```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20])
```

[23]:
```
arr = np.zeros(5) #Crea array de 0
```

[24]:
```
arr
```

[24]:
```
array([0., 0., 0., 0., 0.])
```

[25]:
```
arr = np.linspace(0,1,100) #Cota inferior, cota superior, cantidad partes
```

[26]:
```
arr
```

[26]:
```
array([0.        , 0.01010101, 0.02020202, 0.03030303, 0.04040404,
       0.05050505, 0.06060606, 0.07070707, 0.08080808, 0.09090909,
       0.1010101 , 0.11111111, 0.12121212, 0.13131313, 0.14141414,
       0.15151515, 0.16161616, 0.17171717, 0.18181818, 0.19191919,
       0.2020202 , 0.21212121, 0.22222222, 0.23232323, 0.24242424,
       0.25252525, 0.26262626, 0.27272727, 0.28282828, 0.29292929,
       0.3030303 , 0.31313131, 0.32323232, 0.33333333, 0.34343434,
       0.35353535, 0.36363636, 0.37373737, 0.38383838, 0.39393939,
       0.4040404 , 0.41414141, 0.42424242, 0.43434343, 0.44444444,
       0.45454545, 0.46464646, 0.47474747, 0.48484848, 0.49494949,
       0.50505051, 0.51515152, 0.52525253, 0.53535354, 0.54545455,
       0.55555556, 0.56565657, 0.57575758, 0.58585859, 0.5959596 ,
       0.60606061, 0.61616162, 0.62626263, 0.63636364, 0.64646465,
       0.65656566, 0.66666667, 0.67676768, 0.68686869, 0.6969697 ,
       0.70707071, 0.71717172, 0.72727273, 0.73737374, 0.74747475,
       0.75757576, 0.76767677, 0.77777778, 0.78787879, 0.7979798 ,
       0.80808081, 0.81818182, 0.82828283, 0.83838384, 0.84848485,
       0.85858586, 0.86868687, 0.87878788, 0.88888889, 0.8989899 ,
       0.90909091, 0.91919192, 0.92929293, 0.93939394, 0.94949495,
       0.95959596, 0.96969697, 0.97979798, 0.98989899, 1.        ])
```

[27]:
```
arr = np.logspace(0,1,100, base=10)
#Genera del 0 a 1 (donde log da 0 y donde 1) en escala logaritmica con 100␣
 ↪cortes y en esa base
#Sirve para distribuir mejor los datos (mas lentamente)
```

[28]:
```
arr
```

[28]:
```
array([ 1.        ,  1.02353102,  1.04761575,  1.07226722,  1.09749877,
        1.12332403,  1.149757  ,  1.17681195,  1.20450354,  1.23284674,
        1.26185688,  1.29154967,  1.32194115,  1.35304777,  1.38488637,
```

```
       1.41747416,  1.45082878,  1.48496826,  1.51991108,  1.55567614,
       1.59228279,  1.62975083,  1.66810054,  1.70735265,  1.7475284 ,
       1.78864953,  1.83073828,  1.87381742,  1.91791026,  1.96304065,
       2.009233  ,  2.05651231,  2.10490414,  2.15443469,  2.20513074,
       2.25701972,  2.3101297 ,  2.36448941,  2.42012826,  2.47707636,
       2.53536449,  2.59502421,  2.65608778,  2.71858824,  2.7825594 ,
       2.84803587,  2.91505306,  2.98364724,  3.05385551,  3.12571585,
       3.19926714,  3.27454916,  3.35160265,  3.43046929,  3.51119173,
       3.59381366,  3.67837977,  3.76493581,  3.85352859,  3.94420606,
       4.03701726,  4.1320124 ,  4.22924287,  4.32876128,  4.43062146,
       4.53487851,  4.64158883,  4.75081016,  4.86260158,  4.97702356,
       5.09413801,  5.21400829,  5.33669923,  5.46227722,  5.59081018,
       5.72236766,  5.85702082,  5.9948425 ,  6.13590727,  6.28029144,
       6.42807312,  6.57933225,  6.73415066,  6.8926121 ,  7.05480231,
       7.22080902,  7.39072203,  7.56463328,  7.74263683,  7.92482898,
       8.11130831,  8.30217568,  8.49753436,  8.69749003,  8.90215085,
       9.11162756,  9.32603347,  9.54548457,  9.77009957, 10.        ])
```

[29]:
```
arr2d = np.zeros((5,5)) #(5x5)
```

[30]:
```
arr2d
```

[30]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

[31]:
```
cubo = np.zeros((5,5,5)).astype(int)+1 #Le sumo uno a todo
```

[32]:
```
cubo
```

[32]:
```
array([[[1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1]],

       [[1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1]],

       [[1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
```

```
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1]],

[[1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1]],

[[1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1]]]])
```

[33]:
```
cubo = np.ones((5,5,5)).astype(np.float16)
```

[34]:
```
cubo
```

[34]:
```
array([[[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]],

       [[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]],

       [[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]],

       [[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]],

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

```
       [1., 1., 1., 1., 1.]]], dtype=float16)
```

[35]:
```
cubo.dtype
```

[35]:
```
dtype('float16')
```

[36]:
```
np.empty((2,3,4)) #Revuelve arreglo con basura
```

[36]:
```
array([[[4.66069997e-310, 0.00000000e+000, 4.66069924e-310,
         4.66069924e-310],
        [4.66069924e-310, 4.66069924e-310, 4.66069924e-310,
         4.66069924e-310],
        [4.66069924e-310, 4.66069924e-310, 4.66069924e-310,
         4.66069924e-310]],

       [[4.66069924e-310, 4.66069924e-310, 4.66069924e-310,
         4.66069924e-310],
        [4.66069924e-310, 4.66069924e-310, 4.66069924e-310,
         4.66069924e-310],
        [4.66069924e-310, 4.66069924e-310, 4.66069924e-310,
         4.66069924e-310]]])
```

PELIGRO

¡`np.empty` no devuelve un arreglo de ceros!

[38]:
```
np.eye(4) #Pone 1 en la diagonal
```

[38]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

[40]:
```
np.random.seed(10) #Especifica la semilla para sacar el random (Ya no es tan␣
↪random)
np.random.rand(10)
```

[40]:
```
array([0.77132064, 0.02075195, 0.63364823, 0.74880388, 0.49850701,
       0.22479665, 0.19806286, 0.76053071, 0.16911084, 0.08833981])
```

1.3 Reshaping

[41]:
```
arr = np.arange(1000)
arr
```

[41]:
```
array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
        13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
```

26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38,
39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64,
65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,
91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103,
104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,
156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168,
169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181,
182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194,
195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207,
208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220,
221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233,
234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246,
247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259,
260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272,
273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285,
286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298,
299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311,
312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324,
325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337,
338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350,
351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363,
364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376,
377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389,
390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402,
403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415,
416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428,
429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441,
442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454,
455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467,
468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480,
481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493,
494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506,
507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519,
520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532,
533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545,
546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558,
559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571,
572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584,
585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597,
598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610,
611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623,
624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636,

```
637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649,
650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662,
663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675,
676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688,
689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701,
702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714,
715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727,
728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740,
741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753,
754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766,
767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779,
780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792,
793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805,
806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818,
819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831,
832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844,
845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857,
858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870,
871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883,
884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896,
897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909,
910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922,
923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935,
936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948,
949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961,
962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974,
975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987,
988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999])
```

[42]:
```
arr3d = arr.reshape((10,10,10))
arr3d
```

[42]:
```
array([[[  0,   1,   2,   3,   4,   5,   6,   7,   8,   9],
        [ 10,  11,  12,  13,  14,  15,  16,  17,  18,  19],
        [ 20,  21,  22,  23,  24,  25,  26,  27,  28,  29],
        [ 30,  31,  32,  33,  34,  35,  36,  37,  38,  39],
        [ 40,  41,  42,  43,  44,  45,  46,  47,  48,  49],
        [ 50,  51,  52,  53,  54,  55,  56,  57,  58,  59],
        [ 60,  61,  62,  63,  64,  65,  66,  67,  68,  69],
        [ 70,  71,  72,  73,  74,  75,  76,  77,  78,  79],
        [ 80,  81,  82,  83,  84,  85,  86,  87,  88,  89],
        [ 90,  91,  92,  93,  94,  95,  96,  97,  98,  99]],

       [[100, 101, 102, 103, 104, 105, 106, 107, 108, 109],
        [110, 111, 112, 113, 114, 115, 116, 117, 118, 119],
        [120, 121, 122, 123, 124, 125, 126, 127, 128, 129],
        [130, 131, 132, 133, 134, 135, 136, 137, 138, 139],
```

```
 [140, 141, 142, 143, 144, 145, 146, 147, 148, 149],
 [150, 151, 152, 153, 154, 155, 156, 157, 158, 159],
 [160, 161, 162, 163, 164, 165, 166, 167, 168, 169],
 [170, 171, 172, 173, 174, 175, 176, 177, 178, 179],
 [180, 181, 182, 183, 184, 185, 186, 187, 188, 189],
 [190, 191, 192, 193, 194, 195, 196, 197, 198, 199]],

[[200, 201, 202, 203, 204, 205, 206, 207, 208, 209],
 [210, 211, 212, 213, 214, 215, 216, 217, 218, 219],
 [220, 221, 222, 223, 224, 225, 226, 227, 228, 229],
 [230, 231, 232, 233, 234, 235, 236, 237, 238, 239],
 [240, 241, 242, 243, 244, 245, 246, 247, 248, 249],
 [250, 251, 252, 253, 254, 255, 256, 257, 258, 259],
 [260, 261, 262, 263, 264, 265, 266, 267, 268, 269],
 [270, 271, 272, 273, 274, 275, 276, 277, 278, 279],
 [280, 281, 282, 283, 284, 285, 286, 287, 288, 289],
 [290, 291, 292, 293, 294, 295, 296, 297, 298, 299]],

[[300, 301, 302, 303, 304, 305, 306, 307, 308, 309],
 [310, 311, 312, 313, 314, 315, 316, 317, 318, 319],
 [320, 321, 322, 323, 324, 325, 326, 327, 328, 329],
 [330, 331, 332, 333, 334, 335, 336, 337, 338, 339],
 [340, 341, 342, 343, 344, 345, 346, 347, 348, 349],
 [350, 351, 352, 353, 354, 355, 356, 357, 358, 359],
 [360, 361, 362, 363, 364, 365, 366, 367, 368, 369],
 [370, 371, 372, 373, 374, 375, 376, 377, 378, 379],
 [380, 381, 382, 383, 384, 385, 386, 387, 388, 389],
 [390, 391, 392, 393, 394, 395, 396, 397, 398, 399]],

[[400, 401, 402, 403, 404, 405, 406, 407, 408, 409],
 [410, 411, 412, 413, 414, 415, 416, 417, 418, 419],
 [420, 421, 422, 423, 424, 425, 426, 427, 428, 429],
 [430, 431, 432, 433, 434, 435, 436, 437, 438, 439],
 [440, 441, 442, 443, 444, 445, 446, 447, 448, 449],
 [450, 451, 452, 453, 454, 455, 456, 457, 458, 459],
 [460, 461, 462, 463, 464, 465, 466, 467, 468, 469],
 [470, 471, 472, 473, 474, 475, 476, 477, 478, 479],
 [480, 481, 482, 483, 484, 485, 486, 487, 488, 489],
 [490, 491, 492, 493, 494, 495, 496, 497, 498, 499]],

[[500, 501, 502, 503, 504, 505, 506, 507, 508, 509],
 [510, 511, 512, 513, 514, 515, 516, 517, 518, 519],
 [520, 521, 522, 523, 524, 525, 526, 527, 528, 529],
 [530, 531, 532, 533, 534, 535, 536, 537, 538, 539],
 [540, 541, 542, 543, 544, 545, 546, 547, 548, 549],
 [550, 551, 552, 553, 554, 555, 556, 557, 558, 559],
 [560, 561, 562, 563, 564, 565, 566, 567, 568, 569],
```

```
 [570, 571, 572, 573, 574, 575, 576, 577, 578, 579],
 [580, 581, 582, 583, 584, 585, 586, 587, 588, 589],
 [590, 591, 592, 593, 594, 595, 596, 597, 598, 599]],

[[600, 601, 602, 603, 604, 605, 606, 607, 608, 609],
 [610, 611, 612, 613, 614, 615, 616, 617, 618, 619],
 [620, 621, 622, 623, 624, 625, 626, 627, 628, 629],
 [630, 631, 632, 633, 634, 635, 636, 637, 638, 639],
 [640, 641, 642, 643, 644, 645, 646, 647, 648, 649],
 [650, 651, 652, 653, 654, 655, 656, 657, 658, 659],
 [660, 661, 662, 663, 664, 665, 666, 667, 668, 669],
 [670, 671, 672, 673, 674, 675, 676, 677, 678, 679],
 [680, 681, 682, 683, 684, 685, 686, 687, 688, 689],
 [690, 691, 692, 693, 694, 695, 696, 697, 698, 699]],

[[700, 701, 702, 703, 704, 705, 706, 707, 708, 709],
 [710, 711, 712, 713, 714, 715, 716, 717, 718, 719],
 [720, 721, 722, 723, 724, 725, 726, 727, 728, 729],
 [730, 731, 732, 733, 734, 735, 736, 737, 738, 739],
 [740, 741, 742, 743, 744, 745, 746, 747, 748, 749],
 [750, 751, 752, 753, 754, 755, 756, 757, 758, 759],
 [760, 761, 762, 763, 764, 765, 766, 767, 768, 769],
 [770, 771, 772, 773, 774, 775, 776, 777, 778, 779],
 [780, 781, 782, 783, 784, 785, 786, 787, 788, 789],
 [790, 791, 792, 793, 794, 795, 796, 797, 798, 799]],

[[800, 801, 802, 803, 804, 805, 806, 807, 808, 809],
 [810, 811, 812, 813, 814, 815, 816, 817, 818, 819],
 [820, 821, 822, 823, 824, 825, 826, 827, 828, 829],
 [830, 831, 832, 833, 834, 835, 836, 837, 838, 839],
 [840, 841, 842, 843, 844, 845, 846, 847, 848, 849],
 [850, 851, 852, 853, 854, 855, 856, 857, 858, 859],
 [860, 861, 862, 863, 864, 865, 866, 867, 868, 869],
 [870, 871, 872, 873, 874, 875, 876, 877, 878, 879],
 [880, 881, 882, 883, 884, 885, 886, 887, 888, 889],
 [890, 891, 892, 893, 894, 895, 896, 897, 898, 899]],

[[900, 901, 902, 903, 904, 905, 906, 907, 908, 909],
 [910, 911, 912, 913, 914, 915, 916, 917, 918, 919],
 [920, 921, 922, 923, 924, 925, 926, 927, 928, 929],
 [930, 931, 932, 933, 934, 935, 936, 937, 938, 939],
 [940, 941, 942, 943, 944, 945, 946, 947, 948, 949],
 [950, 951, 952, 953, 954, 955, 956, 957, 958, 959],
 [960, 961, 962, 963, 964, 965, 966, 967, 968, 969],
 [970, 971, 972, 973, 974, 975, 976, 977, 978, 979],
 [980, 981, 982, 983, 984, 985, 986, 987, 988, 989],
 [990, 991, 992, 993, 994, 995, 996, 997, 998, 999]]])
```

[43]:
```
arr3d.ndim
```

[43]:
```
3
```

[44]:
```
arr3d.shape
```

[44]:
```
(10, 10, 10)
```

[45]:
```
arr3d
```

[45]:
```
array([[[  0,   1,   2,   3,   4,   5,   6,   7,   8,   9],
        [ 10,  11,  12,  13,  14,  15,  16,  17,  18,  19],
        [ 20,  21,  22,  23,  24,  25,  26,  27,  28,  29],
        [ 30,  31,  32,  33,  34,  35,  36,  37,  38,  39],
        [ 40,  41,  42,  43,  44,  45,  46,  47,  48,  49],
        [ 50,  51,  52,  53,  54,  55,  56,  57,  58,  59],
        [ 60,  61,  62,  63,  64,  65,  66,  67,  68,  69],
        [ 70,  71,  72,  73,  74,  75,  76,  77,  78,  79],
        [ 80,  81,  82,  83,  84,  85,  86,  87,  88,  89],
        [ 90,  91,  92,  93,  94,  95,  96,  97,  98,  99]],

       [[100, 101, 102, 103, 104, 105, 106, 107, 108, 109],
        [110, 111, 112, 113, 114, 115, 116, 117, 118, 119],
        [120, 121, 122, 123, 124, 125, 126, 127, 128, 129],
        [130, 131, 132, 133, 134, 135, 136, 137, 138, 139],
        [140, 141, 142, 143, 144, 145, 146, 147, 148, 149],
        [150, 151, 152, 153, 154, 155, 156, 157, 158, 159],
        [160, 161, 162, 163, 164, 165, 166, 167, 168, 169],
        [170, 171, 172, 173, 174, 175, 176, 177, 178, 179],
        [180, 181, 182, 183, 184, 185, 186, 187, 188, 189],
        [190, 191, 192, 193, 194, 195, 196, 197, 198, 199]],

       [[200, 201, 202, 203, 204, 205, 206, 207, 208, 209],
        [210, 211, 212, 213, 214, 215, 216, 217, 218, 219],
        [220, 221, 222, 223, 224, 225, 226, 227, 228, 229],
        [230, 231, 232, 233, 234, 235, 236, 237, 238, 239],
        [240, 241, 242, 243, 244, 245, 246, 247, 248, 249],
        [250, 251, 252, 253, 254, 255, 256, 257, 258, 259],
        [260, 261, 262, 263, 264, 265, 266, 267, 268, 269],
        [270, 271, 272, 273, 274, 275, 276, 277, 278, 279],
        [280, 281, 282, 283, 284, 285, 286, 287, 288, 289],
        [290, 291, 292, 293, 294, 295, 296, 297, 298, 299]],

       [[300, 301, 302, 303, 304, 305, 306, 307, 308, 309],
        [310, 311, 312, 313, 314, 315, 316, 317, 318, 319],
        [320, 321, 322, 323, 324, 325, 326, 327, 328, 329],
        [330, 331, 332, 333, 334, 335, 336, 337, 338, 339],
```

```
 [340, 341, 342, 343, 344, 345, 346, 347, 348, 349],
 [350, 351, 352, 353, 354, 355, 356, 357, 358, 359],
 [360, 361, 362, 363, 364, 365, 366, 367, 368, 369],
 [370, 371, 372, 373, 374, 375, 376, 377, 378, 379],
 [380, 381, 382, 383, 384, 385, 386, 387, 388, 389],
 [390, 391, 392, 393, 394, 395, 396, 397, 398, 399]],

[[400, 401, 402, 403, 404, 405, 406, 407, 408, 409],
 [410, 411, 412, 413, 414, 415, 416, 417, 418, 419],
 [420, 421, 422, 423, 424, 425, 426, 427, 428, 429],
 [430, 431, 432, 433, 434, 435, 436, 437, 438, 439],
 [440, 441, 442, 443, 444, 445, 446, 447, 448, 449],
 [450, 451, 452, 453, 454, 455, 456, 457, 458, 459],
 [460, 461, 462, 463, 464, 465, 466, 467, 468, 469],
 [470, 471, 472, 473, 474, 475, 476, 477, 478, 479],
 [480, 481, 482, 483, 484, 485, 486, 487, 488, 489],
 [490, 491, 492, 493, 494, 495, 496, 497, 498, 499]],

[[500, 501, 502, 503, 504, 505, 506, 507, 508, 509],
 [510, 511, 512, 513, 514, 515, 516, 517, 518, 519],
 [520, 521, 522, 523, 524, 525, 526, 527, 528, 529],
 [530, 531, 532, 533, 534, 535, 536, 537, 538, 539],
 [540, 541, 542, 543, 544, 545, 546, 547, 548, 549],
 [550, 551, 552, 553, 554, 555, 556, 557, 558, 559],
 [560, 561, 562, 563, 564, 565, 566, 567, 568, 569],
 [570, 571, 572, 573, 574, 575, 576, 577, 578, 579],
 [580, 581, 582, 583, 584, 585, 586, 587, 588, 589],
 [590, 591, 592, 593, 594, 595, 596, 597, 598, 599]],

[[600, 601, 602, 603, 604, 605, 606, 607, 608, 609],
 [610, 611, 612, 613, 614, 615, 616, 617, 618, 619],
 [620, 621, 622, 623, 624, 625, 626, 627, 628, 629],
 [630, 631, 632, 633, 634, 635, 636, 637, 638, 639],
 [640, 641, 642, 643, 644, 645, 646, 647, 648, 649],
 [650, 651, 652, 653, 654, 655, 656, 657, 658, 659],
 [660, 661, 662, 663, 664, 665, 666, 667, 668, 669],
 [670, 671, 672, 673, 674, 675, 676, 677, 678, 679],
 [680, 681, 682, 683, 684, 685, 686, 687, 688, 689],
 [690, 691, 692, 693, 694, 695, 696, 697, 698, 699]],

[[700, 701, 702, 703, 704, 705, 706, 707, 708, 709],
 [710, 711, 712, 713, 714, 715, 716, 717, 718, 719],
 [720, 721, 722, 723, 724, 725, 726, 727, 728, 729],
 [730, 731, 732, 733, 734, 735, 736, 737, 738, 739],
 [740, 741, 742, 743, 744, 745, 746, 747, 748, 749],
 [750, 751, 752, 753, 754, 755, 756, 757, 758, 759],
 [760, 761, 762, 763, 764, 765, 766, 767, 768, 769],
```

```
 [770, 771, 772, 773, 774, 775, 776, 777, 778, 779],
 [780, 781, 782, 783, 784, 785, 786, 787, 788, 789],
 [790, 791, 792, 793, 794, 795, 796, 797, 798, 799]],

[[800, 801, 802, 803, 804, 805, 806, 807, 808, 809],
 [810, 811, 812, 813, 814, 815, 816, 817, 818, 819],
 [820, 821, 822, 823, 824, 825, 826, 827, 828, 829],
 [830, 831, 832, 833, 834, 835, 836, 837, 838, 839],
 [840, 841, 842, 843, 844, 845, 846, 847, 848, 849],
 [850, 851, 852, 853, 854, 855, 856, 857, 858, 859],
 [860, 861, 862, 863, 864, 865, 866, 867, 868, 869],
 [870, 871, 872, 873, 874, 875, 876, 877, 878, 879],
 [880, 881, 882, 883, 884, 885, 886, 887, 888, 889],
 [890, 891, 892, 893, 894, 895, 896, 897, 898, 899]],

[[900, 901, 902, 903, 904, 905, 906, 907, 908, 909],
 [910, 911, 912, 913, 914, 915, 916, 917, 918, 919],
 [920, 921, 922, 923, 924, 925, 926, 927, 928, 929],
 [930, 931, 932, 933, 934, 935, 936, 937, 938, 939],
 [940, 941, 942, 943, 944, 945, 946, 947, 948, 949],
 [950, 951, 952, 953, 954, 955, 956, 957, 958, 959],
 [960, 961, 962, 963, 964, 965, 966, 967, 968, 969],
 [970, 971, 972, 973, 974, 975, 976, 977, 978, 979],
 [980, 981, 982, 983, 984, 985, 986, 987, 988, 989],
 [990, 991, 992, 993, 994, 995, 996, 997, 998, 999]]])
```

[46]:
```
arr = np.arange(200)
```

[47]:
```
arr2d = arr.reshape((10,20))
```

[48]:
```
arr2d
```

[48]:
```
array([[  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
         13,  14,  15,  16,  17,  18,  19],
       [ 20,  21,  22,  23,  24,  25,  26,  27,  28,  29,  30,  31,  32,
         33,  34,  35,  36,  37,  38,  39],
       [ 40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,  52,
         53,  54,  55,  56,  57,  58,  59],
       [ 60,  61,  62,  63,  64,  65,  66,  67,  68,  69,  70,  71,  72,
         73,  74,  75,  76,  77,  78,  79],
       [ 80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,  91,  92,
         93,  94,  95,  96,  97,  98,  99],
       [100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112,
        113, 114, 115, 116, 117, 118, 119],
       [120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132,
        133, 134, 135, 136, 137, 138, 139],
       [140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152,
```

```
         153, 154, 155, 156, 157, 158, 159],
        [160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172,
         173, 174, 175, 176, 177, 178, 179],
        [180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192,
         193, 194, 195, 196, 197, 198, 199]])
```

1.4 Aplanar

[49]:
```
arr = np.zeros((4,4,4,4))
```

[50]:
```
arr.shape
```

[50]:
```
(4, 4, 4, 4)
```

[51]:
```
arr_plano = arr.ravel() #Lo hace de una sola dimension
```

[52]:
```
arr_plano.shape
```

[52]:
```
(256,)
```

[53]:
```
arr_plano
```

[53]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0.])
```

1.5 *Broadcasting*

Hace operaciones a todos los datos

[54]:
```
data = np.array([1,2,3,4,5])
```

[55]:
```
data
```

[55]:
```
array([1, 2, 3, 4, 5])
```

[56]:
```
data + 1
```

[56]:
```
array([2, 3, 4, 5, 6])
```

[57]:
```
data * 2
```

[57]:
```
array([ 2,  4,  6,  8, 10])
```

[58]:
```
data ** 2
```

[58]:
```
array([ 1,  4,  9, 16, 25])
```

1.6 Transponer

[59]:
```
arr = np.arange(15).reshape((3,5))
```

[60]:
```
arr
```

[60]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

[61]:
```
arr.T
```

[61]:
```
array([[ 0,  5, 10],
       [ 1,  6, 11],
       [ 2,  7, 12],
       [ 3,  8, 13],
       [ 4,  9, 14]])
```

¿Qué pasa en varias dimensiones?

[62]:
```
arr  = np.arange(16).reshape((2,2,4))
```

[63]:
```
arr
```

[63]:
```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7]],

       [[ 8,  9, 10, 11],
        [12, 13, 14, 15]]])
```

[64]:
```
arr.transpose((1,0,2)) #Para decidir como permutarlos
```

```
[64]: array([[[ 0,  1,  2,  3],
              [ 8,  9, 10, 11]],

             [[ 4,  5,  6,  7],
              [12, 13, 14, 15]]])
```

`transpose` recibe una `tupla` de los índices de los ejes y los permuta. `(O_o)`

Ejercicio Diseña un ejemplo multidimensional, donde sea obvia la permutación

1.7 Slicing e Indexado

En el ejercicio vimos que el indexado en `arrays` de 1D es igual que el indexado y *slicing* de las listas de python. ¿Pero que sucede en $n-$dimensiones?

1.7.1 Cuidado al hacer *slicing*

```
[65]: arr = np.arange(10)
```

```
[66]: arr
```

```
[66]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

El *slincing* genera (devuelve) una **vista**, si modificas el =array= original, la **vista** se ve modificada también.

```
[67]: arr_slice =  arr[5:8]
      #Es una vista del objeto, si modificas la vista se modifica el original
```

```
[68]: arr_slice
```

```
[68]: array([5, 6, 7])
```

```
[69]: arr_slice[1]= 12345678
```

```
[70]: arr_slice
```

```
[70]: array([       5, 12345678,        7])
```

```
[71]: arr
```

```
[71]: array([       0,        1,        2,        3,        4,        5,
       12345678,        7,        8,        9])
```

```
[72]: arr_slice[:] = 345
```

```
[73]: arr_slice
```

```
[73]: array([345, 345, 345])
```

[74]:
```
arr
```

[74]:
```
array([  0,   1,   2,   3,   4, 345, 345, 345,   8,   9])
```

[75]:
```
arr2 = np.copy(arr) #Crea una copia en la que no comparte memoria
```

[76]:
```
arr2
```

[76]:
```
array([  0,   1,   2,   3,   4, 345, 345, 345,   8,   9])
```

[77]:
```
arr2[5:8] = [5,6,7]
```

[78]:
```
arr2
```

[78]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[79]:
```
arr
```

[79]:
```
array([  0,   1,   2,   3,   4, 345, 345, 345,   8,   9])
```

[80]:
```
np.may_share_memory(arr, arr_slice) #Para saber si comparten memoria
```

[80]:
```
True
```

[81]:
```
np.may_share_memory(arr, arr2)
```

[81]:
```
False
```

1.7.2 Multidimensional

[82]:
```
arr = np.arange(9)
```

[83]:
```
arr.shape = (3,3) #Es como un reshape
```

[84]:
```
arr
```

[84]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

[85]:
```
arr.ndim
```

[85]:
```
2
```

[86]:
```
arr[2]
```

[86]:
```
array([6, 7, 8])
```

[87]:
```
arr[-1] #ultimo renglon
```

[87]:
```
array([6, 7, 8])
```

[88]:
```
arr[1][1] #El uno del renglon 1
```

[88]:
```
4
```

[89]:
```
arr[1:]
```

[89]:
```
array([[3, 4, 5],
       [6, 7, 8]])
```

[90]:
```
arr[:2]
```

[90]:
```
array([[0, 1, 2],
       [3, 4, 5]])
```

[91]:
```
arr[:1,:2]
```

[91]:
```
array([[0, 1]])
```

[92]:
```
arr[1:,:2]
```

[92]:
```
array([[3, 4],
       [6, 7]])
```

[93]:
```
arr[1,] #Nos da todas las columnas del primer renglon
```

[93]:
```
array([3, 4, 5])
```

[94]:
```
arr[1,:2]
```

[94]:
```
array([3, 4])
```

[95]:
```
arr[1,2:]
```

[95]:
```
array([5])
```

[96]:
```
arr[:,1:] #todos los renglones desde el primero
```

[96]:
```
array([[1, 2],
       [4, 5],
       [7, 8]])
```

[97]:
```
arr[:,:1] #Todos los renglones hasta la 1 columna -1
```

[97]:
```
array([[0],
       [3],
       [6]])
```

Ejercicio:

Explique como funciona el *slicing* n-dimensional.

[98]:
```
arr
```

[98]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

[99]:
```
index = arr > 2 #arreglo booleanos
```

[100]:
```
index
```

[100]:
```
array([[False, False, False],
       [ True,  True,  True],
       [ True,  True,  True]])
```

[101]:
```
arr[index] #devuelve el arreglo dependiendo del booleano
```

[101]:
```
array([3, 4, 5, 6, 7, 8])
```

[102]:
```
arr2 = arr[index]
```

[103]:
```
arr
```

[103]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

[104]:
```
arr2
```

[104]:
```
array([3, 4, 5, 6, 7, 8])
```

Ejercicio: (a) Cree un arreglo de 2D 5×5 lleno de unos. (b) Utilice *slicing* para seleccionar 1 cuadrado alrededor del centro y llénelo con 2s. (c) Utilice *slicing* para seleccionar el centro y asígnele 4. (d) Copie el arreglo. (e) Utilice *slicing* lógico para seleccionar el cuadro interno y asígnele cero. (f) En el cuadro copiado, al centro y al cuadro exterior asígnele 0.

1.8 Fancy Indexing

[105]:
```
arr = np.ones((5,4))
arr
```

[105]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

[106]:
```
for i in range(5):
    arr[i] = i
```

[107]:
```
arr
```

[107]:
```
array([[0., 0., 0., 0.],
       [1., 1., 1., 1.],
       [2., 2., 2., 2.],
       [3., 3., 3., 3.],
       [4., 4., 4., 4.]])
```

[108]:
```
arr[[4,3,1,2]] #le indico que renglon quiero y en que orden
```

[108]:
```
array([[4., 4., 4., 4.],
       [3., 3., 3., 3.],
       [1., 1., 1., 1.],
       [2., 2., 2., 2.]])
```

[109]:
```
arr[[-3,-2,-1]] #El - indica ultimo --> y esto es una nueva matriz (vista)
```

[109]:
```
array([[2., 2., 2., 2.],
       [3., 3., 3., 3.],
       [4., 4., 4., 4.]])
```

¿Puedes explicar que hace el *fancy indexing*?

1.9 Funciones Universales

Las *funciones universales* realizan operaciones elemento por elemento en los arreglos.

[110]:
```
arr = np.arange(10)
```

[111]:
```
arr = -1*arr #Es un broadcasting
```

[112]:
```
arr
```

[112]:
```
array([ 0, -1, -2, -3, -4, -5, -6, -7, -8, -9])
```

[114]:
```
#Es el infinito
np.Inf
```

[114]:
```
inf
```

[115]:
```
arr = np.abs(arr)
```

[116]:
```
arr
```

[116]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[117]:
```
np.sqrt(arr)
```

[117]:
```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])
```

[119]:
```
#Devuelve el signo del arreglo
np.sign(arr)
```

[119]:
```
array([0, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

[120]:
```
np.isfinite(arr) #Saber si es finito
```

[120]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True])
```

[121]:
```
np.logical_not(arr) #Solo en 0 da True
```

[121]:
```
array([ True, False, False, False, False, False, False, False, False,
       False])
```

[122]:
```
arr
```

[122]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[125]:
```
arr = np.random.randn(10)
#Genera 10 numeros aleatorios
```

[126]:
```
arr
```

[126]:
```
array([-1.13660221,  0.13513688,  1.484537  , -1.07980489, -1.97772828,
       -1.7433723 ,  0.26607016,  2.38496733,  1.12369125,  1.67262221])
```

[128]:
```
np.ceil(arr)
#Techo
```

[128]:
```
array([-1.,  1.,  2., -1., -1., -1.,  1.,  3.,  2.,  2.])
```

[130]:
```
np.floor(arr)
#Piso
```

[130]:
```
array([-2.,  0.,  1., -2., -2., -2.,  0.,  2.,  1.,  1.])
```

[131]:
```
np.rint(arr) #Lo redondea
```

[131]:
```
array([-1.,  0.,  1., -1., -2., -2.,  0.,  2.,  1.,  2.])
```

[132]:
```
arr2 = np.ones(10)
```

[133]:
```
np.add(arr, arr2) #Suma los 2 arreglos
```

[133]:
```
array([-0.13660221,  1.13513688,  2.484537  , -0.07980489, -0.97772828,
       -0.7433723 ,  1.26607016,  3.38496733,  2.12369125,  2.67262221])
```

[134]:
```
np.multiply(arr, arr2)
```

[134]:
```
array([-1.13660221,  0.13513688,  1.484537  , -1.07980489, -1.97772828,
       -1.7433723 ,  0.26607016,  2.38496733,  1.12369125,  1.67262221])
```

[135]:
```
np.maximum(arr, arr2)
```

[135]:
```
array([1.        , 1.        , 1.484537  , 1.        , 1.        ,
       1.        , 1.        , 2.38496733, 1.12369125, 1.67262221])
```

[136]:
```
np.logical_and(arr, arr2)
```

[136]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True])
```

1.10 Agregaciones

Funciones que calculan operaciones a lo largo de un eje.

[137]:
```
arr
```

[137]:
```
array([-1.13660221,  0.13513688,  1.484537  , -1.07980489, -1.97772828,
       -1.7433723 ,  0.26607016,  2.38496733,  1.12369125,  1.67262221])
```

[138]:
```
arr.sum()
```

[138]:
```
1.1295171675409819
```

[139]:
```
arr.mean()
```

[139]:
```
0.11295171675409818
```

[140]:
```
arr = np.random.randn(5,4)
```

[141]:
```
arr
```

[141]:
```
array([[ 0.09914922,  1.39799638, -0.27124799,  0.61320418],
       [-0.26731719, -0.54930901,  0.1327083 , -0.47614201],
       [ 1.30847308,  0.19501328,  0.40020999, -0.33763234],
       [ 1.25647226, -0.7319695 ,  0.66023155, -0.35087189],
       [-0.93943336, -0.48933722, -0.80459114, -0.21269764]])
```

[142]:
```
arr.sum() #Suma todo el arreglo
```

[142]:
```
0.6329089430523522
```

[143]:
```
arr.mean()
```

[143]:
```
0.03164544715261761
```

[144]:
```
arr.sum(0) #Lo hace sobre las columnas
```

[144]:
```
array([ 1.45734401, -0.17760608,  0.11731071, -0.7641397 ])
```

[145]:
```
arr = np.arange(10)
```

[146]:
```
arr.cumsum()
```

[146]:
```
array([ 0,  1,  3,  6, 10, 15, 21, 28, 36, 45])
```

[147]:
```
arr.cumprod()
```

[147]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

[148]:
```
arr.reshape(2,5)
```

[148]:
```
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

[149]:
```
arr.cumsum()
```

[149]:
```
array([ 0,  1,  3,  6, 10, 15, 21, 28, 36, 45])
```

[150]:
```
arr > 5
```

[150]:
```
array([False, False, False, False, False, False,  True,  True,  True,
        True])
```

[151]:
```
(arr > 5).sum()
```

[151]:
```
4
```

1.11 Operaciones de conjuntos

[152]:
```
arr
```

[152]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[153]:
```
arr2
```

[153]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

[154]:
```
np.unique(arr2)
```

[154]:
```
array([1.])
```

[155]:
```
np.intersect1d(arr, arr2)
```

[155]:
```
array([1.])
```

Ejercicio Usando la definición de **cuadrados mágicos** crea una función que reciba un arreglo e indique si es o no un cuadrado mágico.

1.12 Casting

[156]:
```
a = np.array([1,2,3])
a
```

[156]:
```
array([1, 2, 3])
```

El tipo de mayor jerarquía define el *cast*

[157]:
```
a + 1.5
#Castea todo a reales (Gana la mayor jerarquia)
```

[157]:
```
array([2.5, 3.5, 4.5])
```

La asignación **no** cambia el tipo del arreglo.

[158]:
```
a.dtype
```

[158]:
```
dtype('int64')
```

[159]:
```
(a + 1.5).dtype
```

[159]:
```
dtype('float64')
```

1.13 Tipos

[165]:
```
np.iinfo(np.int64).max, 2**31 - 1   # Prueba con 8, 16, 32 y 64 bits
#Indica hasta que numero puede representar
```

[165]:
```
(9223372036854775807, 2147483647)
```

Ejercicio ¿Qué pasa con `int`?

[166]:
```
np.finfo(np.float64).max
```

[166]:
```
1.7976931348623157e+308
```

[167]:
```
np.finfo(np.float64)
```

[167]:
```
finfo(resolution=1e-15, min=-1.7976931348623157e+308,
max=1.7976931348623157e+308, dtype=float64)
```

[168]:
```
np.finfo(np.float32).eps #Abajo de ese valor se considera un cero
```

[168]:
```
1.1920929e-07
```

[169]:
```
np.float32(1e-8) + np.float32(1) == 1
```

[169]:
```
True
```

[170]:
```
np.float64(1e-8) + np.float64(1) == 1
```

[170]:
```
False
```

1.14 Estructura de datos

[171]:
```
muestra = np.zeros((6,), dtype=[('codigo', 'S4'),('posicion', float), ('valor',␣
↪float)]) #Indicas el nombre y tipo de dato
#Le pone un tipo de dato a cada dimension
```

[172]:
```
muestra
```

[172]:
```
array([(b'', 0., 0.), (b'', 0., 0.), (b'', 0., 0.), (b'', 0., 0.),
       (b'', 0., 0.), (b'', 0., 0.)],
      dtype=[('codigo', 'S4'), ('posicion', '<f8'), ('valor', '<f8')])
```

[173]:
```
muestra.ndim
```

[173]:
```
1
```

[174]:
```
muestra.shape
```

[174]:
```
(6,)
```

[175]:
```
muestra.dtype.names
```

[175]:
```
('codigo', 'posicion', 'valor')
```

[176]:
```
muestra[:] = [('ALFA',   1, 0.37), ('BETA', 1, 0.11), ('TAU', 1,   0.
↪13),('ALFA', 1.5, 0.37), ('ALFA', 3, 0.11), ('TAU', 1.2, 0.13)]
```

[177]:
```
muestra
```

[177]:
```
array([(b'ALFA', 1. , 0.37), (b'BETA', 1. , 0.11), (b'TAU', 1. , 0.13),
       (b'ALFA', 1.5, 0.37), (b'ALFA', 3. , 0.11), (b'TAU', 1.2, 0.13)],
      dtype=[('codigo', 'S4'), ('posicion', '<f8'), ('valor', '<f8')])
```

[178]:
```
muestra.shape
```

[178]:
```
(6,)
```

[179]:
```
muestra['codigo']
```

[179]:
```
array([b'ALFA', b'BETA', b'TAU', b'ALFA', b'ALFA', b'TAU'], dtype='|S4')
```

[180]:
```
muestra[0]['valor']
```

[180]:
```
0.37
```

[181]:
```
muestra[['codigo', 'valor']]
```

[181]:
```
array([(b'ALFA', 0.37), (b'BETA', 0.11), (b'TAU', 0.13), (b'ALFA', 0.37),
       (b'ALFA', 0.11), (b'TAU', 0.13)],
      dtype={'names':['codigo','valor'], 'formats':['S4','<f8'],
'offsets':[0,12], 'itemsize':20})
```

[182]:
```
muestra[muestra['codigo'] == 'ALFA']
```

[182]:
```
array([], shape=(0, 6),
      dtype=[('codigo', 'S4'), ('posicion', '<f8'), ('valor', '<f8')])
```