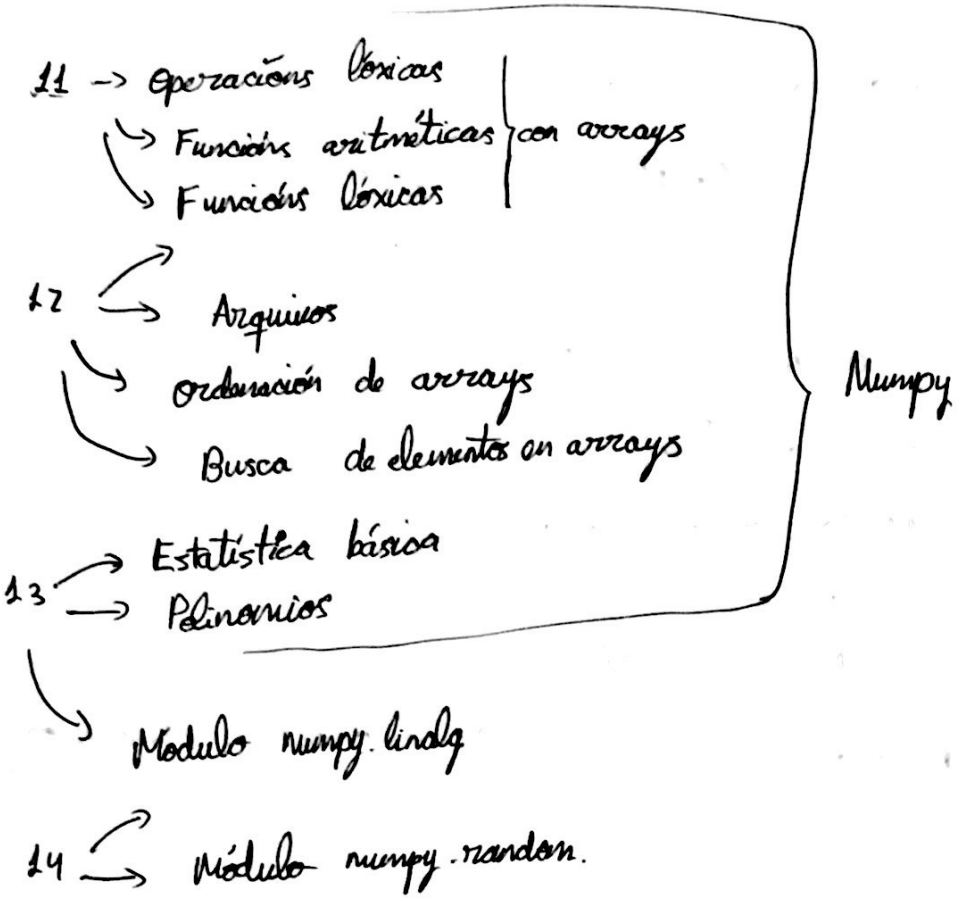
Introducción - Uso do intérprete

→ ipython (abrir int) / Ctrl+D o quit() (para saír)
→ # comentarios
→ %logstart -o nombre.py (gardar en arquivo)
→ historial de comandos (↑ ↓)
→ pwd (ver directorio/ubicación actual)
ls (ver carpetas no directorio actual)
cd (cambiar de directorio)
→ ; (fin de instrucción) → \ (continuación de línea de código)

Índice



Elementos da linguaxe: Variables, ctes, listas, tuplas, diccionarios

variable = expresión ou valor
espazo na memoria do ordenador

⊛ As constantes son un tipo de variables nas que almacenamos valores fixos.

a, b, c = 'ola', 5, True → asignación múltiple

• Tipos de datos: ⊛ A función type() dá o tipo de variable

A) Números:

→ enteiros: int(3) → reais: float(1.2) (flotantes)

→ flotante longo: long(52) → complexos: complex(2+3j)

B) Cadeas de caracteres: str('hola') (string)

ord('1') → devuelve el ASCII de '1'

chr(93) → devuelve el carácter ASCII 93

C) Booleanos (valores léxicos): bool → True / False

Operadores léxicos	O que comproban	
==	igualdade	and (True se ambos son certos)
!= / <>	desigualdade	
>	maior que	or (True se ao menos un é certo)
<	menor que	
>=	maior ou igual	not( ) → inverte o booleano
<=	menor ou igual	

Operadores aritméticos:

+	Suma	2 + 1 ⟶ 3
-	Sustracción	2 - 1 ⟶ 1
	(Tamén negación)	- 5 ⟶ -5
*	Multiplicación	2*3 ⟶ 6
**	Exponenciación	2**3 ⟶ 8
/	División	3.0/2 ⟶ 1.5
		(3/2 ~~⟶~~ 1)
//	División (cociente sen decimales)	3.0/2 ⟶ 1
%	División módulo (Resto)	7 % 2 ⟶ 1

* = Asignación c = a + b

↳ Se engadimos o operador asignación aos anteriores, operarán sobre o valor actual.

* p.e ⟶ c += a ⟶ c = c + a

*

in / not in	x in y / x not in y	Comprueba si x es o no miembro de la secuencia y
is / is not	x is y / x is not y	Comprueba si los id's (id(x)) de ambos elementos son o no iguales

D) **Listas:** Colección ordenadas de datos, que poden ser de distintos tipos.

list( ) lista = [a, 'pomba', [1,2]]

→ Acceso aos elementos da lista:

lista[0] → a

lista[1] → 'pomba'

lista[-1] → [1,2]
↳ último elemento da lista

↳ lista[n-1] (para acceder ao lugar n)

lista[2][0] → 1

Acceso a un elemento dunha lista contida noutra

→ Acceso a sublistas:

a = [1, 2, 3, 4, 5, 6, 7, 8]

a[:] => [1,2,3,4,5,6,7,8]
↓
lista completa

a[2:] => [3,4,5,6,7,8]
↓
a partir do elemento a[2]

a[1:8:2] => [2,4,6,8]
↓ ↓ ↓
inicio (a[1]) fin (8º elemento) distancia dos saltos

a[-1:-4:-2] => [8,6]
↳ cara a atrás

a[:2] => [1,2]
↓
desde o principio ata o elemento en lugar n.

a[2:5] => [3,4,5]
↓
entre o elemento a[2] e o quinto elemento da lista

a[::2] => [1,3,5,7]
↳ Recorre toda a lista de dous en dous elementos

a[::-3] => [8,5,2]
↳ cara a atrás

Funcións sobre listas:

`len(a)` ⟶ número de elementos da lista a (lonxitude)

`max(a)` ⟶ máximo dos elementos da lista a

`min(a)` ⟶ mínimo dos elementos da lista a

`sum(a)` ⟶ suma os elementos da lista a

```
a = [1, 2, 3]      b = [4, 5, 6]
```

`a + b` ⟶ concatena as listas → `[1, 2, 3, 4, 5, 6]`

`a * 2` ⟶ replica a lista → `[1, 2, 3, 1, 2, 3]`

`a[0] = 'ola'` ; `a` ⟶ `['ola', 2, 3]`

Substitúe o elemento da primeira posición por outro dato

`a.insert(x, i)` ⟶ inserta o elemento x na posición i (desprazando un lugar todos os elementos a partir de que ocupaba o lugar i)

`a.append(x)` ⟶ engade o elemento x ao final da lista

`a.index(x)` ⟶ devolve o índice do primeiro elemento da lista a igual a x.

`a.remove(x)` ⟶ elimina o primeiro elemento da lista igual a x

`a.sort()` ⟶ ordea os elementos da lista

`a.reverse()` ⟶ inviste a orde da lista

a.count(x) ⟶ conta o número de veces que aparece x na lista

del a[1] ⟶ elimina o segundo elemento da lista

E) **Tuplas:** Listas de solo lectura (non se poden modificar)

tuple( )

t = (1, 2, 3, 4) ou t = 1, 2, 3, 4

F) **Dicionarios:**

dict( )

Alex = {'nome': 'Alex', 'idade': '18', 'sexo': 'home'}

Alex['sexo'] ⟶ 'home'

Funcións predefinidas:

abs(x) ⟶ valor absoluto de x

cmp(x, y) ⟶ compara x e y
- ⟶ -1 se x < y
- ⟶ 0 se x = y
- ⟶ 1 se x > y

float(x) ⟶ converte un número enteiro a real

int(x) ⟶ converte un número real ou unha cadea a enteiro

type(x) ⟶ devolve o tipo de dato de x

len(s) ⟶ lonxitude dunha secuencia (lista, tupla...)

max(s)
min(s) ⟶ devolven o máximo ou o mínimo da secuencia

tuple(s) ⟶ devolve unha tupla construída cos elementos da secuencia

print x ⟶ visualiza x na consola

str(x) ⟶ transforma x numa cadeia de caracteres

round(x) ⟶ redondea o flotante ao inteiro máis próximo

round(x) ⟶ 3.0

x = 3.4567

round(x,i) ⟶ redondea con i decimais

round(x,2) ⟶ 3.46

range(inicio, fin, incremento) ⟶ lista de números enteiros:

range(2, 8, 2) → [2,4,6]

range(n) ⇒ [0, 1, 2, ..., n-1]

n elementos

range(inicio, fin) ⇒ [inicio, ..., fin-1]

range(2,6) ⇒ [2, 3, 4, 5]

Programas

→ Arquivos con extensión .py que conteñen código fonte en linguaxe python.

Primeiras liñas do arquivo:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
```

Entroda de datos desde o teclado:

raw_input ('texto: ') ⟶ cadea de caracteres que podemos convertir a números usando int() ou float() * float(raw_input())

input ('texto: ') ⟶ le directamente, sen comprobar o tipo de dato.
⊛ permite ler unha lista de números nunha soa sentenza.

Saída de datos por pantalla:

→ Non formateada:

print x ⟶ x

print x, y ⟶ x y

→ Formateada:

%[flags][ancho][.precision]tipo.

- precision → número de decimais
- ancho → número de caracteres
- flags → opciones de rellene
- tipo → tipo de dato

Flags

0 ⟶ rellena con ceros a la izquierda

- ⟶ ajusta el valor escrito a la izquierda

+ ⟶ introduce "+" o "-", dependiendo de si el valor es positivo o negativo.

⟶
¿?

Tipos

- $d_{(i)}$ → enteiro con signo
- u → enteiro sen signo
- o → octal sen signo
- x, X → hexadecimal sen signo
- f → real
- e → real (formato exponencial)
- g → real (formato e con nº de cifras significativas concreto)
- c → carácter
- s → cadea de caracteres

print ("Resultado final: %12.5f" %a) → 0.00004

print ("Resultado final: %012.5f" %a) → 0000.0000365

Caracteres non printables

- \\ → \
- \n → Avance de línea
- \r → retorno de carro
- \t → tabulador

Ficheiros

- Abrir arquivo: `f = open('arquivo.txt', 'modo')`
  - `'arquivo.txt'` → ruta do arquivo
  - `'modo'` → uso
    - `'r'` (só lectura)
    - `'w'` (escribir desde o principio)
    - `'a'` (escribir ó final do arquivo)
    - `'r+'` (ler e escribir)
  - `f` → obxecto de tipo file

- Pechar arquivo: `f.close()`

- Posicionamento no ficheiro
  - `f.tell()` → devolve un enteiro coa posición actual de lectura / escritura.
  - `f.seek(bytes)` → move a posición actual a *bytes* desde o comezo do arquivo
    - *bytes* → nº de byte

- Lectura:
  - `f.read()` → le todo o arquivo
  - `f.read(bytes)` → le o número de bytes indicado
  - `f.readline()` → le unha liña (ata que atope un fin de liña `'\n'`)
  - `f.readlines()` → le todas as liñas do arquivo
    - ↳ crea unha lista
  - A partir da posición actual
  - para separar os datos nunha lista: `list=linha.split()`

- Escritura:
  - `f.write('cadea')` → escribe unha cadea de caracteres
  - `f.writelines(secuencia)` → escribe cada elemento da secuencia nunha liña
    - ↳ p.e. lista

✳
- `f.closed` → devolve verdadeiro se o arquivo está pechado
- `f.mode` → devolve o modo no que está aberto o arquivo
- `f.name` → devolve o nome do arquivo
- `f.encoding` → devolve a codificación de caracteres do arquivo

✳ Excepcions →
```
try:
except IOError:
       ValueError
```

Control de Flujo

* Indentación ⟶ "sangría" ⟶ 4 espazos en branco
  ↳ Obrigatoria para implementar estruturas de fluxo

→ Estruturas de control:
  ↓ → expresións
  ↓
  selección / iteración
  (condicionais)

• Codificación: indica é comezo a codificación de caracteres empregada no arquivo
  ↳ (non evitar) caracteres estraños como a ñ

```
# -*- coding: utf-8 -*-
```

Sentenzas de selección ⟶ Operadores relacionais ou lóxicos

• Bloque (if)

```
if expresión :
    Liña 1
    Liña 2
Liña 3
```

→ Se a expresión é Verdadeira execútase o bloque de dentro do if e despois a Liña 3

→ Se a expresión é Falsa, omítese o bloque de código dentro do if e execútase directamente a liña 3.

• Bloque (if : else)

```
if expresión :
    }B1
else :
    } B2
```

⟶ Se a expresión é verdadeira execútase o bloque B1, se é falsa, o bloque B2.

• Bloque (if : dif : else)

```
if expresion1 :
    }B1
dif expresion 2:
    } B2
( else :
    } BA )
```

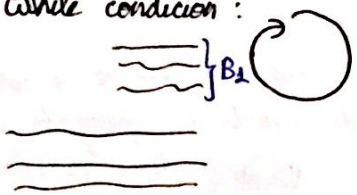
⟶ Avalíanse as expresións por orde ata atopar unha verdadeira, cuxo bloque se executa (e remata a selección).

Se ningunha expresión é verdadeira execútase o bloque de código do else.

Sentenzas de iteración (bucles ou lazos):

→ Permiten executar repetidamente o mesmo código
- → mentres se cumpla unha condición
- → un certo número de veces

cuhile condición :



→ Mentres se cumpre a condición execútase o bloque B_1 unha e outra vez.

As sentenzas de B_1 deben modificar as variábeis que interveñen na condición para que deixe de cumprirse nalgún momento.

for x in secuencia:
- → a variable toma os valores da secuencia.
- → range(n) → a variable toma valores desde 0 ata n-1
  range (inicio, fin, incremento)

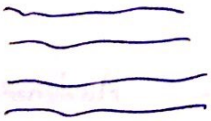
⊛ modificacións do fluxo →
- break → sae do bucle cuhile ou for
- continue → salta ó inicio do bucle ignorando o resto do código e volvendo a avaliar a expresión

---

Definición de funcions →
- Parámetros de entrada
- Valores de salida

} Interfaz

→ def nombrefunción(a, b,):



} Variábles locales

return s, d,

Módulos

- Biblioteca estándar
- instalables (Internet)
- propios

Acceso:
- → `import math` / `math.sin(0)`
- → `from math import sin` / `sin(0)`
- → `from math import *` / `sin(0)`

□ Módulo time → medida do tempo de execución dun bloque de código

`clock()` → mide o tempo actual

Módulo math → funcións matemáticas para números reais:

`sin()`, `cos()`, `tan()`, `asin()`, `acos()`, `atan()`

`sinh()`, `cosh()`, `tanh()`, `asinh()`, `acosh()`, `atanh()`

`degrees()` → transforma de radiáns a graos

`radians()` → transforma de graos a radiáns

`pi`, `e` (ctes)

`sqrt()` → raíz cadrada

`exp()`, `log()` (ln), `log10()` ($\log_{10}$)

`floor()` → redondeo por defecto (float)
`ceil()` → redondeo por exceso (float)
`round()` → redondeo ao enteiro máis próximo (float)
`fabs()` → valor absoluto

`trunc()` → parte enteira (int)

`factorial()` → factorial (int > 0)

`isnan()`, `isinf()` → Comproba se é: Not a Number / Infinito

`max(, , , )`, `min(, , , )` → máximo e mínimo dos argumentos

`choice(secuencia)` → Devolve un elemento aleatorio da secuencia

from math import *

Matplotlib.pyplot

↳ representación gráfica

⊛ interactivamente → (ipython --pylab)

(matplotlib.pyplot + numpy)

from matplotlib.pyplot import *

plot (x , y , 'fmt')

- x: vector do eixo x
- y: vector do eixo y
- 'fmt': tipo de liña/marcas p.e. 'b*-'

de faltar este argumento usa un vector cos elementos [0, 1, ..., N-1]

1. cor: 'b' azul, 'r' verm., 'm' ...
3. liña: '-' continua, '--' discontinua, ':' punteada
2. marcas: 'o' círculo, 'v' triángulo, 'D' diamante, '*' asterisco, 's' cadrado

subplot (filas, columnas, celda) → crea unha matriz de gráficos e activa celda desa matriz para facer nela un gráfico.

semilogx(x, y) , semilogy(x,y) , loglog(x,y) → escala logarítmica nos eixos indicados.

polar (theta , r, fmt) → gráfico en coordenadas polares
- theta: ángulos
- r: módulos
- fmt: tipo de gráfico

bar (x,y) → gráfico de barras (vertical)

barh (x , y) → gráfico de barras (horizontal)
- x: posición das barras
- y: dimensión das barras

pie (x) → gráfico de tarta

hist(x) → histograma (representa a distribución das frecuencias con que aparecen os elementos de x)

fill (x , alpha)

plot_surface(x,y,z) → gráfico en 3D
↳ Di que non está definido

scatter (x, y) → gráfico disperso

⊛ show(False)

from matplotlib.pyplot import *

savefig('nome.extensión')
↓
png, pdf, ps, eps, svg

Propiedades do gráfico: plot(..., label='...')
↳ lenda do gráfico

title('título'), xlabel('lenda no eixo x'), ylabel('lenda no eixo y')
, fontsize = 14, color = 'red')

from matplotlib.pyplot import *

axis([xmin, xmax, ymin, ymax]) → fixa a escala dos eixos

ylim() xlim() → devolven os límites mínimo e máximo de cada eixo

clf() → limpa o contido da figura

grid(True) grid(False) → pon/quita unha reixa sobre o gráfico

hold(True) hold(False) → activa/desactiva o mantemento do gráfico ao representar outros

xticks() = locs, labels
yticks() = locs, labels
} posicións e textos das etiquetas dos eixos.

xticks(range(n), ('a', 'da', ...,))
↳ nº de separacións ↳ etiquetas
↳ arange(n) → importando numpy

legend(loc=)
↓
posición da lenda

'upper right' 'upper left'
'lower right' 'lower left' 'center'
'center right' 'center left'
'lower center' 'upper center'

plot(..., label='texto')

Numpy

array ⟶ como unha lista na que todos os elementos son dun mesmo tipo.

- cadeas (|Sn)
- booleanos (bool)
- enteiros con signo (int) 32/64
- enteiros sen signo (uint) 32/64
- reais (float) 32/64
- complexos (complex) 128/64

from numpy import *

Propiedades dos array: (a é un array)

- a.shape ⟶ tupla coas dimensións
- a.ndim ⟶ número de dimensións
- a.size ⟶ número de elementos
  - size(a, (1)) ↳ eixo
- a.itemsize ⟶ tamaño dos elementos en bytes
- a.nbytes ⟶ tamaño total en bytes
- a.dtype ⟶ tipo de dato dos elementos

(*) Cambiar o tipo de datos dun array ⟶ (b=)array(a, dtype=)

Crear arrays:

(a =) array(secuencia, 'tipodedato') ⟶ p.e. a = array([1,2,3,4,5,6], 'int32')

b = array([[1,2,3],[4,5,6]], 'int64')
"matriz"

(a =) arange(inicio, fin, incremento, dtype =) ⟶ permite incrementos decimais
↳ o fin non se inclúe no array

(a =) linspace(inicio, fin, num = , endpoint = T/F, retstep = T/F)

- num = número de puntos equiespaciados
- endpoint = T/F: True → inclúe o punto final; False → non inclúe o punto final
- retstep = T/F: True → mostra ao final do array o incremento entre os puntos.

(a =) ones(shape, dtype =) ⟶ crea unha matriz de uns (ou vector)

ones(5) ⟶ array([1., 1., 1., 1., 1.])

ones([2,3]) ⟶ array([[1., 1., 1.,
1., 1., 1.]])

(a =) zeros(shape, dtype =) ⟶ crea unha matriz (ou vector) con ceros (igual completamente)

Copia de arrays:

→ Copia por referencia: `b = a`

comparten o mesmo espazo na memoria ⇒ se se modifica un tamén se modifica o outro

→ Copia por valor: `b = copy(a)` → copia real

`b = a.copy()`

Acceso a elementos dos arrays:

`a[posición]`

`a[inicio : fin]` → o primeiro índice é o 0 e non colle o elemento fin.

- índices negativos empezan polo final
- se non existe o índice pedido → IndexError

`a [[pos, pos, ...]]` → acceso a unha lista (array) de elementos do array.

`a [0, 0]` → 1 elemento dun array de dimensión 2 (tantos índices como dimensións).

`a[0, :]` → todos os índices posibles nesa dimensión

`(a =) a.reshape(newshape)` → modificar as dimensións dos arrays (ten que coincidir a dimensión global)

- 6
- [2, 3]
- ...

* import numpy from

Modificar arrays:

(a=) a.reshape(newshape)

(a=) insert(array(a), posición(s), valor(es), (axis=(None))?) → inserta os valores nas posicións

(a=) append(array(a), valor(es), (axis=(None))) → inserta os valores ao final do array

(a=) delete(array(a), posición(s), (axis=(None))) → devolve un array sen os elementos nas posicións

(a=) a.flatten(order='C'/'F') → transforma a dimensión 1 (vecto

conversión por filas

conversión por columnas

(a=) ravel(a, order='C'/'F') → transforma a dimensión 1 (vecto

(c=) concatenate((a,b), axis=) → teñen que ter a mesma forma

↳ dimensión na que se unen 0 → a / b ; 1 → a b

(a=) a.T → transposta do array

(c=) hstack([a,b]) → engade columnas a b

vstack([a,b]) → engade filas a / b

from numpy import *

Operacións aritméticas (array (a) e número (x))

(entre arrays → dimensións idénticas → operacións elemento a elemento)

+ , - , * , /

a/x ⇒ array($\left[\frac{a_0}{x}, \frac{a_1}{x}, \ldots, \frac{a_n}{x}\right]$)

a*x ⇒ array($[a_0*x, a_1*x, \ldots, a_n*x]$)

a - x ⇒ array($[a_0-x, a_1-x, \ldots, a_n-x]$)

a*b = array($[a_0*b_0, \ldots,$ …

Operacións relacionais:

Comparan arrays ou arrays e datos e devolven arrays de booleanos.

$>$, $>=$, $<$, $<=$, $==$, $!=$

p.e. a = array([1,2,3]) ; b = array([2,1,4])

a < b ⇒ array([True, False, True])

a == 2 ⇒ array([False, True, False])

Operacións lóxicas

Entre datos de tipo booleano ⇒ array booleano

↳ opéranse elemento a elemento

&, |, ~

↳ and ↳ or ↳ Not → contrario (só un operando)

↳ True se True en algún

↳ True só se True en ambos

from numpy import *

Funcións aritméticas → usan vectores como argumentos, aplicándose a todos os elementos do vector.

Trigonométricas: sin( ), cos( ), tan( ), arcsin( ), arccos( ), arctan( )

degrees( ) ↳ convierte a grados , radians( ) ↳ convierte a rad ,

Hiperbólicas: sinh( ), cosh( ), tanh( ), arcsinh( ), arctanh( )

Redondeo: round_(array (a), n) ↳ número de decimais redondeo

→ redondea a enteiro

rint( ) = fix( )

floor( ), ceil( ), trunc( ) ↳ parte enteira

Exp e log: exp() log() log10() log2()

sumas, productos e diferencias:

prod(array (a)) → produto de todos os elementos

prod(array, axis =) → 1 → produtos de cada fila; 0 → produtos de cada columna

sum(array)

sum(array, axis =)

↳ se se trata dun array de booleanos obtemos o número de "True"

cumprod(array(, axis =))

cumsum(array(, axis =))

} vai multiplicando / sumando os elementos sucesivamente, cada elemento é operado co resultado da operación anterior. (pódese facer globalmente ou por filas / columnas)

cross(a, b) → ¿?

sqrt() power() ?

from numpy import *

Funcións lóxicas

all(array (a)) → True se todos os elementos do array son non nulos

all(array (a), axis)
- 1 → columna a columna
- 0 → fila a fila

any(array (a)(, axis)) → True se hai algún elemento non nulo

alldose(a, b) → True sé los dos arrays tienen todos los elementos iguales (deben ter a mesma forma)

array_equal(a,b) → True se teñen a mesma forma e elementos.

greater(a, b) → devolve nun array o resultado da comparación elemento a elemento ($a_i > b_i$)

greater_equal(a, b) → o mesmo pero $a_i >= b_i$

arrays de comparacións elemento a elemento

less (a,b)	(<)
less_equal (a,b)	(<=)
not_equal (a,b)	(!=)
equal (a,b)	(==)
greater (a,b)	(>)
greater_equal (a,b)	(>=)

logical_and/or/not ( ) → dúas condicións
(a,b) (a,b) (a)
→ Para cada elemento ou par de elementos:
True <=> x ≠ 0 . False <=> x = 0

((not) x in z) → condición
elemento → lista

Entrada / saída → Arquivos

savetxt ('nomearquivo.txt', a, (formato = '%(ancho_columna).(numdecimais)tipo de dato'))
a → array

loadtxt ('nomearquivo', dtype =)
'int'
'float'

Ordenación de arrays

sort (array, (axis =)) → ordea de menor a maior

argsort (array, (axis =)) → dá un array cas posicións ás que cambiarían os elementos do array ao ordenalo.

* podes ver numpy.org

Procura de elementos en arrays

`cubere (condición)` → devolve un ~~array~~ cas posicións onde se cumple a condición
p.e. (a < 3)

En realidade devolve unha tupla con un array para cada coordenada (das posicións se queremos o array `cubere(condición)[0]`

(*) `from matplotlib.mlab import *`
`find(condición)` → Devolve un array cas posicións onde se cumple

`cubere (condición, a, b)` → Elemento de a onde a condición sexa certa ou elemento de b onde sexa falsa.

`extract (condición, a)` → devolve un array (dimensión 1) cos elementos do array onde a condición é verdadeira.

`from numpy import *`

`(nan) argmax ( a , (axis =) )` → devolve a/s posición/s dos valores do array iguais ao valor máximo

ignorar os valores NaN

0
1 → devolve un array ca posición do elemento máximo de cada columna (0) ou fila (1).

`(nan) argmin (a , (axis =))` → "

`nonzero (a)` → tupla de arrays coas posicións dos elementos non nulos en cada dimensión.

`flatnonzero (a)` → array de unha dimensión coas posicións dos elementos non nulos.

`isnan(a)` → True nas posicións onde haxa un NaN (array)

`isinf(a)` → True nas posicións onde hai valores ∞. (array)

`from numpy import *`

Estatística básica → En todas estas funcións se pode engadir un segundo argumento "axis ="

amin (a)
amax (a)
nanmin (a)
nanmax (a)
ptp (a) → "peak to peak" → devolve máximo − mínimo

std (a) → desviación típica
var (a) → varianza

mean (a, (axis , weights))
↳ array con os pesos de cada elemento para a media ponderada.

median (a)

Manipulación de polinomios

Polinomios → vectores dos seus coeficientes
↳ p = array ([$coef_n$, $coef_{n-1}$, ..., $coef_2$, $coef_1$, $coef_0$])
↳ coeficiente do monomio máis significativo na posición 0
↳ 0 no lugar dos monomios que non existen.

polyval (p, x) → avalía o polinomio p no punto x

poly ([secuencia de ceros]) → devolve un array os coeficientes dun polinomio cos primeiros m coeficientes nulos e $coef_{m+1} = 1$
↳ m 0's

p.e. poly ([0,0,0]) ⟹ array ([1,0,0,0])

roots (p) → devolve un array coas raíces do polinomio

polyder (p, m) → devolve a m-ésima derivada do polinomio

polyint (p, (m, k)) → devolve un valor da integral indefinida (m-ésima)
↳ constante a sumar ⊛ por defecto m = 1, k = 0

from numpy import *

polyadd(p,q) → suma polinomios polysub(p,q) → diferencia (p-q)

polydiv(p,q) → devolve unha tupla con dous arrays, o cociente (1º) e o resto (2º) da división $\frac{p}{q}$

polymul(p,q) → devolve o produto dos polinomios p e q

?

polyfit(x, y, deg((, rcond, full, w, cov)))

↳ grado do polinomio

↳ Axusta os puntos dos vectores x e y a un polinomio de grao deg usando o método dos mínimos cadrados.

↳ Devolve os coficientes do polinomio

⊛ minimiza a suma dos erros cadráticos entre (x_i, y_i) e $(x_i, p(x_i))$

$$E = \sum_{i=1}^{n} (y_i - p(x_i))^2$$

⊛ Axustes de función exp., pot, log e recíproca (p. 43-46)

from numpy import *

---

from numpy.linalg import *

Álxebra lineal a → array

!!!

det(a) → determinante de a

trace(a) → suma dos elementos da diagonal principal de a

diag(a) → devolve num array a diagonal principal de a

triu(a) → (⋱)
tril(a) → (⋰)
→ Matriz (array) triangular superior (triu) ou inferior (tril)

diagflat(a) → Matriz (array) diagonal (co array a na diagonal)

↳ se dim(a) > 1 → convierte a a dim 1 e crea

from numpy.linalg import

!!

inv(a) → matriz inversa de a

!!

dot(a,b) → produto das matrices a e b (array)

↳ se son dous vectores (1D) fai o produto escalar

!!

matrix_power(a,n) → potencia dunha matriz cadrada a elevada a un enteiro n (a^n)

solve(a,b) → resolve un sistema de ecuacións lineais $ax = b$

a: matriz coeficientes
b: termo independente
↳ ten que ser cadrada¿?

matrix_rank(a) → rango da matriz

eig(a) → Devolve unha tupla con dous arrays, cos autovalores e os autovectores da matriz a.

Erro → LinAlgError

Números aleatorios → Submódulo random

from numpy.random import *

rand() → número aleatorio no intervalo [0,1)

rand($d_0, d_1, \ldots, d_n$) → array das dimensións indicadas con números aleatorios entre 0 e 1.

random(shape) → crea un array de números aleatorios coa forma indicada. Se é de máis dunha dimensión, debe indicarse a forma nunha lista.

seed() → xera unha semente

shuffle(a) → baralla a secuencia ou array a, e sobrescribe a co resultado

permutation(x) → permutacións aleatorias en x:

→ Se x é un enteiro realiza permutacións sobre arange(x)
→ Se x é un vector devolve o vector baralhado (non sobrescribe)
→ Se x é unha secuencia → permuta na 1ª dimensión

Interpolación lineal ← `from numpy import *`

coordenadas x e y dos puntos coñecidos. (vectores)

$y_2 = interp(x_2, x, y)$ ← array de valores (linspace/arange)

↳ función interpelante ⇒ $y_2(x_2)$

Cálculo simbólico → `from sympy import *`

Definición de variables simbólicas: (declaración)

```
x = symbols('x')
```

`limit(f, variable, p)` — f: función; p: punto

`diff(f, variable, p)` — f: función; p: punto

`integrate(f, variable)` → Integral indefinida

`integrate(f, (variable, a, b))` → Integral definida no intervalo [a, b]

`solve(función, variable)` → Resolve a ecuación f(x) = 0

`f.series(x, c, orde+1).evalf()` — variable; orde do polinomio; c: punto no que se avoléa

↳ polinomio de Taylor

`Sum(an, (n, valor inicial, ∞)).evalf()` → Suma da serie $\sum a_n$ (valor final)

`sympify(expresión)`

`lambdify(symbols expresión)` → función lambda