INTRODUCIÓN A PROGRAMACIÓN EN PYTHON

As mulleres na programación e na informática

Características de python

- Linguaxe interpretado. Linguaxe simple e expresiva.
- Pódese utilizar nun entorno interactivo, que nos permite facer probas e aclarar dúbidas das características da linguaxe.
- Permite a programación imperativa e orientada a obxectos.
- Ten moitas estructuras de datos incorporadas a linguaxe.
- Ten gran variedade de bibliotecas (*libraries*).
- O código pódese agrupar en módulos e paquetes.
- Permite manexo de excepcións.

Empezando con Python

- Invocación do intérprete
  - **`python3`**
- Sair do intérprete
  - Control-D (Linux), Control-Z (Windows) ou **`quit()`**
- Comentarios en python: desde que aparece o caracter “#” ate o final da liña.
- Despois da invocación ó intérprete, pódese ver (**`>>>`**) que indica que o intérprete está esperando comandos do usuario.

Uso de python

https://docs.python.org/3/library/index.html#library-index

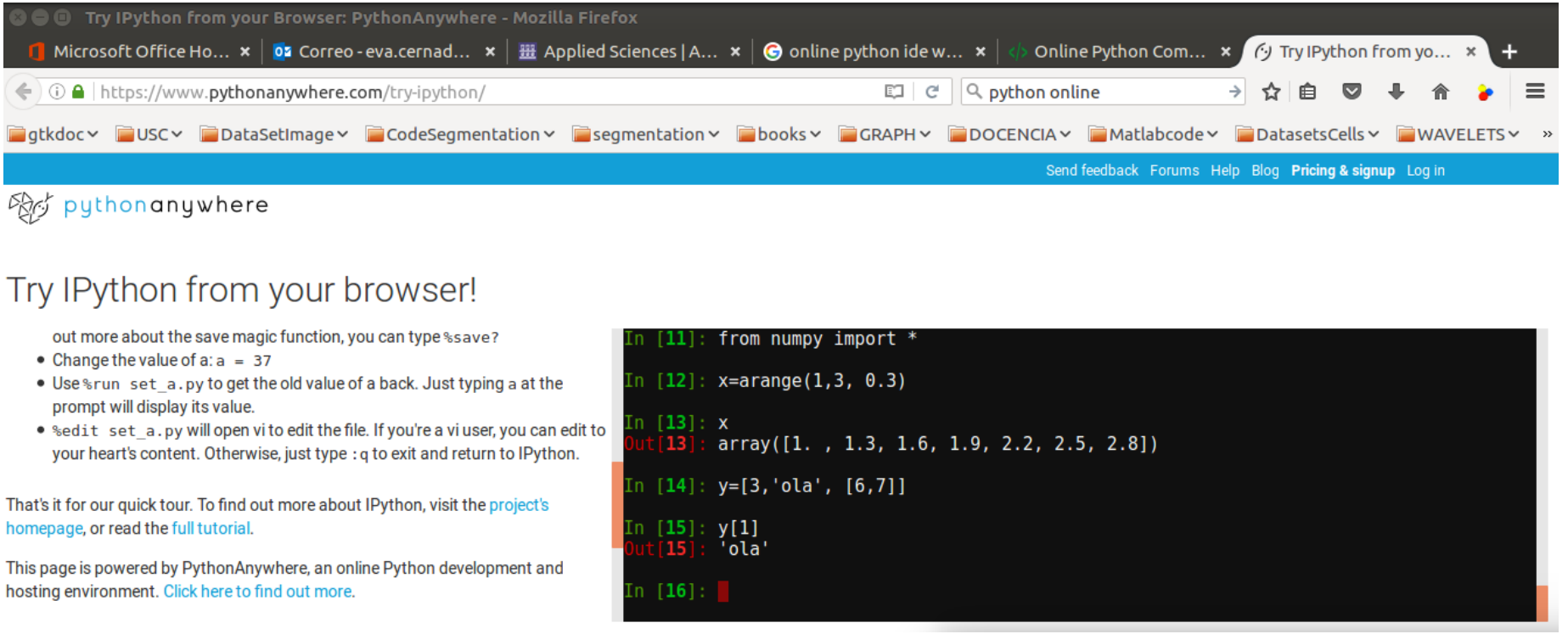
- Dúas formas de executar código python:
  - **Sesión interactiva**: introducir sentenzas no intérprete e obter unha resposta do intérprete para cada liña.
  - Escribir o código dun programa e executalo.
- Nas sesións interactivas podemos utilizar **`ipython3`**: consola interactiva de python con autocompletado de código e axuda.
  - Executando o comando **`%logstart -o sesion1.py`** cando se entra en **`ipython3`** garda no arquivo **`sesion1.py`** os comando que se executan e a súa saída.

Algúns trucos de ipython

- Pódese navegar polo historial de comandos como en Linux: utilizando a frecha para arriba e para abaixo.
- Pódese utilizar os comandos de Linux tipo: `cd, ls, pwd`, ...

Ipython online

- Podes executar este interprete de Python online nun navegador: https://www.pythonanywhere.com/try-ipython/



Algúns elementos da linguaxe Python

- Variables
- Constantes
- Listas
- Tuplas
- Diccionarios

Variables

- **VARIABLE** : é un espazo para almacenar datos modificables na memoria do ordenador.
- Para asignar un valor a unha variable hai que executar a sentenza:

variable = expresion ou valor

- O nome da variable consta de letras minúsculas, maiúsculas, díxitos e o símbolo de subraiado (_), sempre que o díxito non estea no comezo do nome.
- Tampouco se pode usar como nomes de variables as palabras reservadas da linguaxe python.
- Python distingue entre maiúsculas e minúsculas. As variables z e Z non son a mesma.
- Os nomes das variables deben describir o dato que representan para que os programas sexan máis lexibles.

Constantes

- **CONSTANTE**: é un tipo de variable que se utiliza para almacenar valores fixos.
- Recoméndase utilizar nomes descritivos en maiúscula e separando palabras cos guións baixos.
  - Exemplo: **MINHA_CONSTANTE = 15**
- Asignación múltiple: tanto para variables como constantes, pódese asignar valor a múltiples variables nunha soa instrucción:
  - `>>> a, b, c = 'string', 15, True`

Tipos de datos básicos

- Números: enteiros, reais, complexos
  - >>> **x=5** # esto é un enteiro
  - >>> **y=24.4** # esto é un real
  - >>> **z=3+5j** # esto é un número complexo
  - >>> # a funcion **type**(variable) devolve o tipo de variable.
- Cadeas de caracteres

  >>> **c="Ola a todas"** # cadea de caracteres
- Valores booleanos (lóxicos): **True** (certo) e **False** (falso) .

Operadores aritméticos

Son operadores binarios ou unarios que devolven un número.

Operador	**Descrición**	**Exemplo**
+	Suma	r=2 + 1 # r vale 3
-	Resta	r=6 - 4 # r vale 2
-	Negación	r=-5 # r vale -5
*****	Multiplicación	r=3 * 4 # r vale 12
****	Exponenciación	r=2 ** 3 # r vale 8
/	División	r=3.5 / 2 # r vale 1.75
//	División enteira	r=3.5 // 2 # r vale 1.0
%	Módulo (resto da división)	r= 7 % 2 # r vale 1

Listas (I)

- Unha lista é unha colección ordenada de datos de tipo: número, cadea caracteres, lista, etc.
- Crear unha lista (entre corchetes e separando os elementos con coma):
  - `>>> minhalista=[22, True, “unha lista”, [1, 2]]`
- Acceder a cada elemento da lista (lista[indice]): Os índices comezan en 0 (para empezar a contar polo principio) e números negativos (comeza a contar polo final con -1 o primeiro elemento):
  - `>>> v1 = minhalista[0]` # v1 vale 22
  - `>>> v2 = minhalista[-1]` # v2 vale [1, 2]
- Acceder a un elemento dunha lista contida noutra lista:
  - `>>> v3 = minhalista[3][0]` # v3 vale 1

Listas (II)

- Acceder a unha sublista (lista[indice-inicio : indice-fin]:
  - >>> **`v4=minhalista[1:3]`** # v4 vale [True, “unha lista”]
  - >>> **`v5=minhalista[:]`** # v5 vale minhalista
  - >>> **`v6=minhalista[:2]`** # v6 vale [22, True]
- Acceder a unha sublista con elementos alternos (lista[inicio:fin:salto]) cada salto elementos:
  - >>> **`ml=[3,4,6,9,2,1,6,8]`** # crear lista ml
  - >>> **`v1=ml[1:8:3]`** # v1 vale [4,2,8]
  - >>> **`v2=ml[-1:-8:-2]`** ou **`v2=ml[::-2]`** # v2 vale [8, 1, 9, 4]
- Modificar ó tamaño dunha lista:
  - >>> **`ml[1:7]=[True]`** # ml vale [3, True, 8]

Listas (III)

- Lonxitude dunha lista: **`len(lista)`**
- Funcións sobra lista:
  - `>>> ml=[3,4,6,9,2,1,6,8]` # crear lista ml
  - `>>> len(ml)` # número de elementos de ml
  - `>>> max(ml)` # máximo dos elementos de ml
  - `>>> min(ml)` # mínimo dos elementos de ml
  - `>>>sum(ml)` # suma os elementos de ml
- Operadores + e * con listas:
  - `>>> ml2=[3, 2, 4]` # crear lista ml2
  - `>>> ml + ml2` # concatena listas [3, 4, 6, 9, 2, 1, 6, 8, 3, 2, 4]
  - `>>> [1,2]*2` # replica a lista [1, 2, 1, 2]

Listas (IV)

- Métodos úteis para listas (https://docs.python.org/3/tutorial/datastructures.html:
  - **`insert(i,x)`** inserta o elemento x na posición i (despois da inserción x ocupa a posición i).
  - **`append(x)`** engade o elemento x ó final da lista.
  - **`index(x)`** devolve o índice do primeiro elemento da lista igual a x.
  - **`remove(x)`** elimina o primeiro elemento da lista igual a x.
  - **`sort()`** ordea os elementos da lista.
  - **`reverse()`** invirte a orde dos elementos da lista.
  - **`count(x)`** conta o número de veces que aparece x na lista.
  - Para eliminar un elemento da lista utilizando o seu índice utilizar: **`del nomelista[indice]`**.

Listas (V)

- Exemplos:
  - `>>> x = [2,6,1,4]`
  - `>>> x.sort()` # devolve [1, 2, 4, 6]
  - `>>> x.remove(4)` # devolve [1, 2, 6]
  - `>>> x.insert(0, 10)` # devolve [10, 1, 2, 6]
  - `>>> x.append(20)` # devolve [10, 1, 2, 6, 20]
  - `>>> x.index(6)` # devolve 3
  - `>>> x.append(6)` # devolve [10, 1, 2, 6, 20, 6]
  - `>>> x.count(6)` # devolve 2
  - `>>> del x[4]` # devolve [10, 1, 2, 6, 6]

Tuplas (I)

- As tuplas son tipos de datos semellantes ás listas pero o seu tamaño é fixo, non se poden modificar e créanse dun xeito diferente.
- Creación de tuplas:
  - `>>> minhatupla=1,2,3,4,5,6` ou `minhatupla=(1,2,3,4,5,6)`

Algunhas funcións predefinidas

- **`abs(x)`**: valor absoluto de x
- **`chr(i)`**: devolve o carácter que ten código ASCII i.
- **`float(x)`**: convirte un número enteiro a real (float).
- **`int(x)`**: convirte un número ou unha cadea a enteiro.
- **`long(x)`**: igual que int(x) pero convirte a un enteiro longo.
- **`type(x)`**: devolve o tipo de dato de x.
- **`len(s)`**: lonxitude dunha secuencia (list, tuple, ...).

Algunhas funcións predefinidas

- **`max(s)`**: devolve o máximo da secuencia.
- **`min(s)`**: devolve o mínimo da secuencia.
- **`tuple(s)`**: devolve unha tupla construida cos elementos dunha secuencia.
- **`print(x)`**: visualiza na consola x.
- **`str(x)`**: devolve unha cadea que representa a x de xeito “lexible”.
- **`round(x)`**: redondea ó número flotante máis próximo. Pode usarse con un (número sen decimais) ou dous argumentos (número con tantos decimais como indica o segundo argumento).

Algunhas funcións predefinidas

- **`range(inicio, fin , incremento)`**: crea unha lista con números enteiros. Pódese utilizar con un (lista de números consecutivos entre 0 e fin>0), dous (lista de números entre inicio e fin--(inicio<fin)) ou tres (lista de números entre inicio e fin separados incremento) argumentos. Para visualizar o seu contido utiliza a función **list()** Exemplos:
  - `>>> list(range(4))` # devolve [0, 1, 2, 3]
  - `>>> list(range(2, 6))` # devolve [2, 3, 4, 5]
  - `>>> list(range(-2, 2))` # devolve [-2, -1, 0, 1]
  - `>>> list(range(-2, 10, 3))` # devolve [-2, 1, 4, 7]
  - `>>> list(range(10, -2, -3))` # devolve [10, 7, 4, 1]