

Tema 0 - Programación básica con Python

Curso de Física Computacional

M. en C. Gustavo Contreras Mayén

Contenido

- 1 ¿Qué necesitamos para trabajar el curso?
- 2 Introducción
 - Tipado dinámico
 - Fuertemente tipado
 - Programación orientada a objetos
 - Complementos para Python
- 3 Iniciando con Python
- 4 Operadores relacionales
- 5 Variables
- 6 Listas, Tuplas y Diccionarios
 - Listas
 - Tuplas
 - Diccionarios
- 7 Función Range()
- 8 Funciones intrínsecas

¿Qué necesitamos para trabajar el curso?

El curso requiere como herramienta un equipo de cómputo y software, con la ventaja de que el software está disponible bajo la licencia GNU y podemos descargarlo en versiones tanto para Linux como para Windows.

Es conveniente que configures tu equipo de casa o personal antes de iniciar el trabajo, ya que cuando tenemos todo disponible, no tendremos inconvenientes posteriores de funcionamiento.

Software necesario para el trabajo del curso

- Python
- Spyder2
- matplotlib
- gnuplot
- Visual Python

Python como lenguaje de programación

La versión que usaremos es la 2.7.x, aunque ya existe la versión 3, tenemos mucho más documentación de la 2.7, si instalan de manera completa una distribución de linux, lo más seguro es que les instale por defecto, la 3, pero podemos tener instaladas las dos versiones, sin problemas de compatibilidad.

La liga para descargar es:

<http://www.python.org/>

Versión integrada de Python

Existe una versión de pago por un paquete integrado con Python que reúne la mayoría de librerías que vamos a utilizar, la ventaja es que no tendremos que instalar posteriormente más allá de Visual Python.

El sitio es

<https://www.enthought.com/products/epd/>

Entorno de desarrollo integrado

La ventaja de usar un IDLE (Interface Development Environment) es que en un sólo espacio de trabajo, contamos con un editor de texto, documentación de ayuda, consola de ejecución y un soporte bastante útil en donde se resaltan las instrucciones propias del lenguaje, así como valores numéricos y de texto; en ese mismo espacio podemos ejecutar el código sin necesidad de abrir una terminal adicional.

Spyder2 es el IDLE enfocado al manejo de Python, podemos descargarlo de su sitio:

<http://code.google.com/p/spyderlib/>

Componer Python con librerías

Si no descargamos la distribución de Enthought, nos veremos en la necesidad de descargar librerías adicionales para nuestro curso. Una de ellas es el graficador matplotlib, que está disponible en el sitio:

<http://matplotlib.org/>

Una ventaja es que en cada sitio, la documentación y foros de apoyo se actualizan constantemente y por tanto, tener una ayuda con el software de manera directa.

Cuando tenemos el conjunto solución mediante un algoritmo, nuestro trabajo no termina ahí, como hemos aprendido en la carrera, debemos de visualizar ese conjunto de datos y dar una interpretación de los mismos. Existen bastantes programas tanto de licencia como GNU que permiten hacer esta tarea, el más cómodo es gnuplot, podemos conseguirlo desde:

<http://www.gnuplot.info/>

Extendiendo la visualización con Python

En algunos temas del curso, habrá la necesidad de generar no sólo gráficas más complejas, sino que es posible también crear animaciones con Python, aunque este punto merece de por sí un tiempo para aprenderlo, veremos algunas cosas básicas, ya quedarán con el gusto para continuar el trabajo y practicar de manera externa. La liga para descargar la librería es:

<http://www.vpython.org/>

Contenido

1 ¿Qué necesitamos para trabajar el curso?

2 **Introducción**

- Tipado dinámico
- Fuertemente tipado
- Programación orientada a objetos
- Complementos para Python

3 Iniciando con Python

4 Operadores relacionales

5 Variables

6 Listas, Tuplas y Diccionarios

- Listas
- Tuplas
- Diccionarios

7 Función Range()

8 Funciones intrínsecas

Para el curso de Física Computacional será necesario que usemos un lenguaje de programación para apoyarnos en la solución de los problemas y algoritmos.

El lenguaje de nuestra elección es un medio para alcanzar nuestro objetivo del curso, más no el fin, por lo que revisaremos lo más básico de Python, dando la oportunidad de que por tu cuenta, logres un mayor conocimiento y práctica con Python.



- Lenguaje de programación de alto nivel, interpretado.



- Lenguaje de programación de alto nivel, interpretado.
- Desarrollado por Guido van Rossum a principios de los años 90.



- Lenguaje de programación de alto nivel, interpretado.
- Desarrollado por Guido van Rossum a principios de los años 90.
- Es multiplataforma (UNIX, Solaris, Linux, DOS, Windows, OS/2, Mac OS, etc.)



- Lenguaje de programación de alto nivel, interpretado.
- Desarrollado por Guido van Rossum a principios de los años 90.
- Es multiplataforma (UNIX, Solaris, Linux, DOS, Windows, OS/2, Mac OS, etc.)
- Software libre: Python Software Foundation License (PSFL)



- Lenguaje de programación de alto nivel, interpretado.
- Desarrollado por Guido van Rossum a principios de los años 90.
- Es multiplataforma (UNIX, Solaris, Linux, DOS, Windows, OS/2, Mac OS, etc.)
- Software libre: Python Software Foundation License (PSFL)
- Tipado dinámico.



- Lenguaje de programación de alto nivel, interpretado.
- Desarrollado por Guido van Rossum a principios de los años 90.
- Es multiplataforma (UNIX, Solaris, Linux, DOS, Windows, OS/2, Mac OS, etc.)
- Software libre: Python Software Foundation License (PSFL)
- Tipado dinámico.
- Fuertemente tipado.



- Lenguaje de programación de alto nivel, interpretado.
- Desarrollado por Guido van Rossum a principios de los años 90.
- Es multiplataforma (UNIX, Solaris, Linux, DOS, Windows, OS/2, Mac OS, etc.)
- Software libre: Python Software Foundation License (PSFL)
- Tipado dinámico.
- Fuertemente tipado.
- Orientado a objetos.

Tipado dinámico

Cada dato es de un tipo determinado y sólo se puede operar con él de formas bien definidas.

La ventaja es que **NO hay que declarar variables antes de su uso.**

Fuertemente tipado

Se dice que es un lenguaje cuyos tipos son estrictos. Java y Python son fuertemente tipados.

Si tiene un tipo de dato entero, no puede tratarlo como una cadena de texto sin convertirlo explícitamente.

Programación orientada a objetos

La programación orientada a objetos es un paradigma de programación que busca representar entidades u objetos agrupando datos y métodos que puedan describir sus características y comportamientos.

Complementos para Python

- NumPy: paquete fundamental para computación científica.

Complementos para Python

- NumPy: paquete fundamental para computación científica.
- SciPy: librería para computación científica (extiende a NumPy)

Complementos para Python

- NumPy: paquete fundamental para computación científica.
- SciPy: librería para computación científica (extiende a NumPy)
- matplotlib: librería para gráficos 2D (soporta gráficos 3D también)

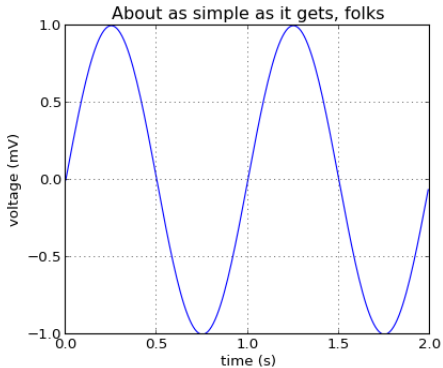
Complementos para Python

- NumPy: paquete fundamental para computación científica.
- SciPy: librería para computación científica (extiende a NumPy)
- matplotlib: librería para gráficos 2D (soporta gráficos 3D también)
- Mayavi: librería para gráficos y visualización de datos 3D.

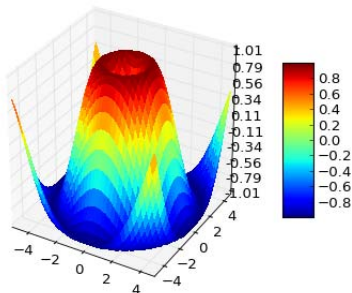
Complementos para Python

- NumPy: paquete fundamental para computación científica.
- SciPy: librería para computación científica (extiende a NumPy)
- matplotlib: librería para gráficos 2D (soporta gráficos 3D también)
- Mayavi: librería para gráficos y visualización de datos 3D.
- iPython: consola interactiva para Python.

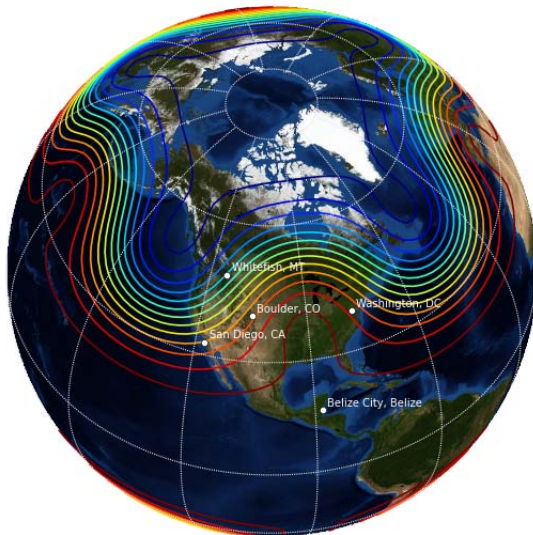
Ejemplos de gráficas



Ejemplos de gráficas



Ejemplos de gráficas



Contenido

- 1 ¿Qué necesitamos para trabajar el curso?
- 2 Introducción
 - Tipado dinámico
 - Fuertemente tipado
 - Programación orientada a objetos
 - Complementos para Python
- 3 **Iniciando con Python**
- 4 Operadores relacionales
- 5 Variables
- 6 Listas, Tuplas y Diccionarios
 - Listas
 - Tuplas
 - Diccionarios
- 7 Función Range()
- 8 Funciones intrínsecas

Consola de trabajo en Python

Hay dos modos de trabajo en Python, cada uno de ellos depende de nuestra habilidad:

- **Modo rudo:** trabajo directo en la consola.

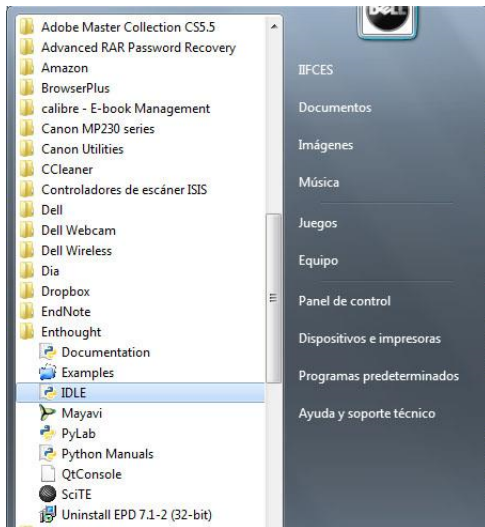
Consola de trabajo en Python

Hay dos modos de trabajo en Python, cada uno de ellos depende de nuestra habilidad:

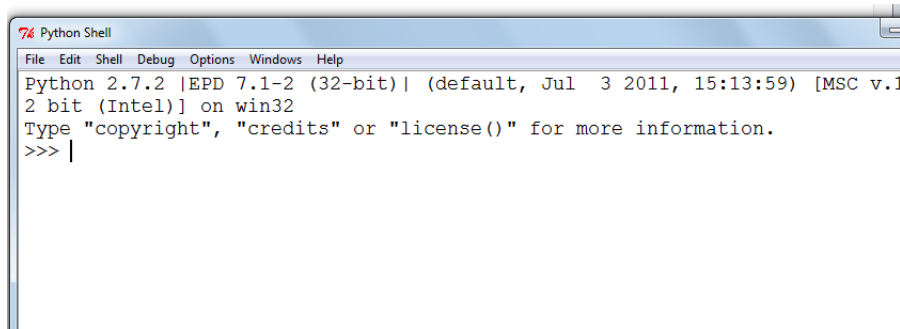
- **Modo rudo:** trabajo directo en la consola.
- **Modo amigable:** a través de una interface IDLE (Entorno de Desarrollo Integrado)

Modo Rudo

Para los usuarios en Windows, hay que localizar la carpeta de Enthought y de ahí seleccionamos IDLE.



Lo que nos abrirá una *terminal* en donde podemos iniciar el trabajo con Python.



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.7.2 |EPD 7.1-2 (32-bit)| (default, Jul  3 2011, 15:13:59) [MSC v.1
2 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> |
```

Para los usuarios con linux

Abrimos una terminal y tecleamos python, presionamos Enter y el prompt de la terminal cambia dejándonos la apariencia de >>>, adicionalmente vemos la versión de trabajo.

```
gustavo@Gus-Acer:~$ python
Python 2.7.3 (default, Sep 26 2012, 21:53:58)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```



Python como calculadora

Una vez abierta la sesión en Python, podemos aprovechar al máximo Python, una de las primeras facilidades que tenemos, es que contamos con una calculadora a la mano, sólo hay que ir escribiendo las operaciones.

Operadores aritméticos

```
>>> 3+4
```

Operadores aritméticos

```
>>> 3+4
```

```
7
```


Operadores aritméticos

```
>>> 3+4
```

```
7
```

```
>>> 3/4
```

Operadores aritméticos

```
>>> 3+4
```

```
7
```

```
>>> 3/4
```

```
0
```

Operadores aritméticos

```
>>> 3+4
```

```
7
```

```
>>> 3/4
```

```
0
```

```
>>> 3.0/4.0
```

Operadores aritméticos

```
>>> 3+4  
7
```

```
>>> 3/4  
0
```

```
>>> 3.0/4.0  
0.75
```

Operadores aritméticos

```
>>> 3+4  
7
```

```
>>> 3/4  
0
```

```
>>> 3.0/4.0  
0.75
```

```
>>> 5.0 / 10 * 2 + 5
```

Operadores aritméticos

```
>>> 3+4  
7
```

```
>>> 3/4  
0
```

```
>>> 3.0/4.0  
0.75
```

```
>>> 5.0 / 10 * 2 + 5  
6
```

```
>>> 5.0 / (10 * 2 + 5)
```

```
>>> 5.0 / (10 * 2 + 5)  
0.2
```



```
>>> 5.0 / (10 * 2 + 5)  
0.2
```

```
>>> 2**3**2
```

```
>>> 5.0 / (10 * 2 + 5)  
0.2
```

```
>>> 2**3**2  
512
```

```
>>> 5.0 / (10 * 2 + 5)  
0.2
```

```
>>> 2**3**2  
512
```

```
>>> (2**3)**2
```

```
>>> 5.0 / (10 * 2 + 5)  
0.2
```

```
>>> 2**3**2  
512
```

```
>>> (2**3)**2  
64
```

```
>>> 5.0 / (10 * 2 + 5)  
0.2
```

```
>>> 2**3**2  
512
```

```
>>> (2**3)**2  
64
```

```
>>> 17%3%2
```

```
>>> 5.0 / (10 * 2 + 5)  
0.2
```

```
>>> 2**3**2  
512
```

```
>>> (2**3)**2  
64
```

```
>>> 17%3%2  
0
```

Tabla de operadores

Operador	Operación	Ejemplo	Resultado
**	Potencia	$2 * 3$	8
*	Multiplicación	$7 * 3$	21
/	División	$10.5 / 2$	5.25
//	División entera	$10.5 // 2$	5.0
+	Suma	$3 + 4$	7
-	Resta	$6 - 8$	-2
%	Módulo	$15 \% 6$	3

Precedencia de operadores 1

- 1 Las expresiones contenidas dentro de pares de paréntesis son evaluadas primero. En el caso de expresiones con paréntesis anidados, los operadores en el par de paréntesis más interno son aplicados primero.

Precedencia de operadores 1

- 1 Las expresiones contenidas dentro de pares de paréntesis son evaluadas primero. En el caso de expresiones con paréntesis anidados, los operadores en el par de paréntesis más interno son aplicados primero.
- 2 Las operaciones de exponentes son aplicadas después. Si una expresión contiene muchas operaciones de exponentes, los operadores son aplicados de derecha a izquierda.

Precedencia de operadores 2

- 3 La multiplicación, división y módulo son las siguientes en ser aplicadas. Si una expresión contiene muchas multiplicaciones, divisiones u operaciones de módulo, los operadores se aplican de izquierda a derecha.

Precedencia de operadores 2

- 3 La multiplicación, división y módulo son las siguientes en ser aplicadas. Si una expresión contiene muchas multiplicaciones, divisiones u operaciones de módulo, los operadores se aplican de izquierda a derecha.
- 4 Suma y resta son las operaciones que se aplican por último. Si una expresión contiene muchas operaciones de suma y resta, los operadores son aplicados de izquierda a derecha. La suma y resta tienen el mismo nivel de precedencia.

Contenido

- 1 ¿Qué necesitamos para trabajar el curso?
- 2 Introducción
 - Tipado dinámico
 - Fuertemente tipado
 - Programación orientada a objetos
 - Complementos para Python
- 3 Iniciando con Python
- 4 Operadores relacionales
- 5 Variables
- 6 Listas, Tuplas y Diccionarios
 - Listas
 - Tuplas
 - Diccionarios
- 7 Función Range()
- 8 Funciones intrínsecas

Operadores relacionales (de comparación)

Tipos de datos lógicos: False (0) y True (1)

```
1+2>7-3
```

Operadores relacionales (de comparación)

Tipos de datos lógicos: False (0) y True (1)

```
1+2>7-3  
False
```

Operadores relacionales (de comparación)

Tipos de datos lógicos: False (0) y True (1)

```
1+2>7-3
```

```
False
```

```
1<2<3
```

Operadores relacionales (de comparación)

Tipos de datos lógicos: False (0) y True (1)

```
1+2>7-3
```

```
False
```

```
1<2<3
```

```
True
```


Operadores relacionales (de comparación)

Tipos de datos lógicos: False (0) y True (1)

```
1+2>7-3
```

```
False
```

```
1<2<3
```

```
True
```

```
1>2==2<3
```

Operadores relacionales (de comparación)

Tipos de datos lógicos: False (0) y True (1)

```
1+2>7-3
```

```
False
```

```
1<2<3
```

```
True
```

```
1>2==2<3
```

```
False
```

Operadores relacionales (de comparación)

Tipos de datos lógicos: False (0) y True (1)

```
1+2>7-3
```

```
False
```

```
1<2<3
```

```
True
```

```
1>2==2<3
```

```
False
```

```
1>(2==2)<3
```

Operadores relacionales (de comparación)

Tipos de datos lógicos: False (0) y True (1)

```
1+2>7-3
```

```
False
```

```
1<2<3
```

```
True
```

```
1>2==2<3
```

```
False
```

```
1>(2==2)<3
```

```
False
```

$3 > 4 < 5$

```
3>4<5  
False
```

```
3>4<5
```

```
False
```

```
1.0/3<0.33333
```

```
3>4<5
```

```
False
```

```
1.0/3<0.33333
```

```
False
```



```
3>4<5
```

```
False
```

```
1.0/3<0.33333
```

```
False
```

```
5.0/3>=11/7.0
```

```
3>4<5
```

```
False
```

```
1.0/3<0.33333
```

```
False
```

```
5.0/3>=11/7.0
```

```
3>4<5
```

```
False
```

```
1.0/3<0.33333
```

```
False
```

```
5.0/3>=11/7.0
```

```
3>4<5
```

```
False
```

```
1.0/3<0.33333
```

```
False
```

```
5.0/3>=11/7.0
```

```
True
```

```
3>4<5  
False
```

```
1.0/3<0.33333  
False
```

```
5.0/3>=11/7.0  
True
```

```
2**(2./3)<3**(3./4)
```

```
3>4<5
```

```
False
```

```
1.0/3<0.33333
```

```
False
```

```
5.0/3>=11/7.0
```

```
True
```

```
2**(2./3)<3**(3./4)
```

```
True
```

Tabla de operadores relacionales

Operador	Operación	Ejemplo	Resultado
==	Igual a	4 == 5	False
!=	Diferente de	2 != 3	True
<	Menor que	10 < 4	False
>	Mayor que	5 > -4	True
>=	Menor o igual que	7 <= 7	True
>=	Mayor o igual que	3.5 >= 10	False

Operadores lógicos (booleanos)

Operador	Operación	Ejemplo	Resultado
and	conjunción	False and True	False
or	disyunción	False or True	True
not	negación	not True	False

Tabla de verdad

A	B	A and B	A or B	not A
True	True	True	True	True
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Tipos de datos

Cada lenguaje de programación requiere de un conjunto de tipos de datos para operar, cada uno está caracterizado por un nombre, un tamaño de espacio en memoria y un intervalo.

Realizar operaciones entre diferentes tipos de datos nos va a generar un error, ya que como hemos mencionado, Python es un lenguaje fuertemente tipado, moraleja: sumar peras con peras y manzanas con manzanas.

Tabla de tipos de datos

Tipo	Descripción	bits	Rango	Ejemplo
bool	booleano	8	sin rango	True o False
int	entero	16	$[-2^{15}, 2^{15} - 1]$	327
long int	entero largo	32	$[0, 2^{32} - 1]$	24334253234L
float	real (punto flotante)	32	$[-2^{31}, 2^{31} - 1]$	3.1416
string	string (cadena)	32	$[-2^{31}, 2^{31} - 1]$	'hola'
tuple	tupla	32	$[3.4 \times 10^{-38}, 3.4 \times 10^{38}]$	1, 'aja', 2.0)
list	lista	64	$[1.7 \times 10^{-308}, 1.7 \times 10^{308}]$	[1, 'aja', 2.0]
dict	diccionario	80	$[3.4 \times 10^{-4932}, 3.4 \times 10^{4932}]$	'a':7.0, 23: Tr

Palabras reservadas

No se pueden utilizar dentro del código

and	assert	break	class	continue	def
del	elif	else	except	exec	finally
for	from	global	if	import	in
is	lambda	not	or	pass	print
raise	return	try	while	yield	del

Identificadores

Son nombres que hacen referencia a los objetos que componen un programa: constantes, variables, funciones, etc.

Reglas para construir identificadores:

- 1 El primer carácter debe ser una letra o el carácter de subrayado (guión bajo)

Identificadores

Son nombres que hacen referencia a los objetos que componen un programa: constantes, variables, funciones, etc.

Reglas para construir identificadores:

- 1 El primer carácter debe ser una letra o el carácter de subrayado (guión bajo)
- 2 El primer carácter puede ir seguido de un número variable de dígitos numéricos, letras o caracteres de subrayado.

Identificadores

Son nombres que hacen referencia a los objetos que componen un programa: constantes, variables, funciones, etc.

Reglas para construir identificadores:

- 1 El primer carácter debe ser una letra o el carácter de subrayado (guión bajo)
- 2 El primer carácter puede ir seguido de un número variable de dígitos numéricos, letras o caracteres de subrayado.
- 3 No pueden utilizarse espacios en blanco, ni símbolos de puntuación.

Identificadores

Son nombres que hacen referencia a los objetos que componen un programa: constantes, variables, funciones, etc.

Reglas para construir identificadores:

- 1 El primer carácter debe ser una letra o el carácter de subrayado (guión bajo)
- 2 El primer carácter puede ir seguido de un número variable de dígitos numéricos, letras o caracteres de subrayado.
- 3 No pueden utilizarse espacios en blanco, ni símbolos de puntuación.
- 4 Python distingue mayúsculas y minúsculas.

Identificadores

Son nombres que hacen referencia a los objetos que componen un programa: constantes, variables, funciones, etc.

Reglas para construir identificadores:

- 1 El primer carácter debe ser una letra o el carácter de subrayado (guión bajo)
- 2 El primer carácter puede ir seguido de un número variable de dígitos numéricos, letras o caracteres de subrayado.
- 3 No pueden utilizarse espacios en blanco, ni símbolos de puntuación.
- 4 Python distingue mayúsculas y minúsculas.
- 5 No pueden utilizarse palabras reservadas del lenguaje.

Contenido

- 1 ¿Qué necesitamos para trabajar el curso?
- 2 Introducción
 - Tipado dinámico
 - Fuertemente tipado
 - Programación orientada a objetos
 - Complementos para Python
- 3 Iniciando con Python
- 4 Operadores relacionales
- 5 Variables
- 6 Listas, Tuplas y Diccionarios
 - Listas
 - Tuplas
 - Diccionarios
- 7 Función Range()
- 8 Funciones intrínsecas

Variables

```
>>> base = 2
```

Variables

```
>>> base = 2
```

```
>>> print base
```

Variables

```
>>> base = 2
```

```
>>> print base  
2
```

```
>>> print "base"  
base
```

Variables

```
>>> base = 2
```

```
>>> print base  
2
```

```
>>> print "base"  
base
```

```
>>> base = base + 1
```

```
>>> base
```

```
>>> base
```

```
3
```



```
>>> base
```

```
3
```

```
>>> alt = 4
```

```
>>> base  
3
```

```
>>> alt = 4
```

```
>>> area = base*alt; a= 3
```

```
>>> base  
3
```

```
>>> alt = 4
```

```
>>> area = base*alt; a= 3
```

```
>>> a= 2*a
```

```
>>> area== 2*a
```

```
>>> area== 2*a  
True
```

```
>>> area== 2*a  
True
```

```
>>> x= "uno"; y= "dos"
```

```
>>> area== 2*a  
True
```

```
>>> x= "uno"; y= "dos"
```

```
>>> x
```

```
>>> area== 2*a  
True
```

```
>>> x= "uno"; y= "dos"
```

```
>>> x  
uno
```



```
>>> area== 2*a  
True
```

```
>>> x= "uno"; y= "dos"
```

```
>>> x  
uno
```

```
>>> print x
```

```
>>> area== 2*a  
True
```

```
>>> x= "uno"; y= "dos"
```

```
>>> x  
uno
```

```
>>> print x  
uno
```

```
>>> x+y
```

```
>>> x+y  
unodos
```

```
>>> x+y  
unodos
```

```
>>> print x+y
```

```
>>> x+y  
unodos
```

```
>>> print x+y  
unodos
```

Contenido

- 1 ¿Qué necesitamos para trabajar el curso?
- 2 Introducción
 - Tipado dinámico
 - Fuertemente tipado
 - Programación orientada a objetos
 - Complementos para Python
- 3 Iniciando con Python
- 4 Operadores relacionales
- 5 Variables
- 6 Listas, Tuplas y Diccionarios
 - Listas
 - Tuplas
 - Diccionarios
- 7 Función Range()
- 8 Funciones intrínsecas

Una lista en Python es un contenedor dinámico que puede contener un número ilimitado de elementos. Los elementos se pueden agregar y quitar de la lista, y los elementos no tienen por que ser del mismo tipo. En otras palabras, una lista puede contener elementos de diferentes tipos.

Para declarar una lista en Python, usamos los corchetes `[]` en torno a una lista de los objetos, o vacío entre corchetes para declarar una lista vacía:

```
mi_lista = []
```


Ejercicios con listas

```
milista=[a,"hola",3.0,True]
```

Ejercicios con listas

```
milista=[a,"hola",3.0,True]
```

```
milista
```

Ejercicios con listas

```
milista=[a,"hola",3.0,True]
```

```
milista  
[8,"hola",3.0,True]
```

Ejercicios con listas

```
milista=[a,"hola",3.0,True]
```

```
milista  
[8,"hola",3.0,True]
```

```
milista[0]
```

Ejercicios con listas

```
milista=[a,"hola",3.0,True]
```

```
milista  
[8,"hola",3.0,True]
```

```
milista[0]  
8
```

Ejercicios con listas

```
milista=[a,"hola",3.0,True]
```

```
milista  
[8,"hola",3.0,True]
```

```
milista[0]  
8
```

```
milista[1]
```

Ejercicios con listas

```
milista=[a,"hola",3.0,True]
```

```
milista  
[8,"hola",3.0,True]
```

```
milista[0]  
8
```

```
milista[1]  
'hola'
```

Ejercicios con listas

```
milista=[a,"hola",3.0,True]
```

```
milista  
[8,"hola",3.0,True]
```

```
milista[0]  
8
```

```
milista[1]  
'hola'
```

```
milista[2]
```


Ejercicios con listas

```
milista=[a,"hola",3.0,True]
```

```
milista  
[8,"hola",3.0,True]
```

```
milista[0]  
8
```

```
milista[1]  
'hola'
```

```
milista[2]  
3.0
```

```
milista[1:3]
```

```
milista[1:3]  
'hola',3.0
```

```
milista[1:3]  
'hola',3.0
```

```
milista[0] = 2.0
```

```
milista[1:3]  
'hola',3.0
```

```
milista[0] = 2.0
```

```
milista
```

```
milista[1:3]  
'hola',3.0
```

```
milista[0] = 2.0
```

```
milista  
[2.0,"hola",3.0,True]
```

```
milista[1:3]  
'hola',3.0
```

```
milista[0] = 2.0
```

```
milista  
[2.0,"hola",3.0,True]
```

```
milista[-1]
```

```
milista[1:3]  
'hola',3.0
```

```
milista[0] = 2.0
```

```
milista  
[2.0,"hola",3.0,True]
```

```
milista[-1]  
True
```



```
milista.append("otro")
```

```
milista.append("otro")
```

```
milista
```

```
milista.append("otro")
```

```
milista  
[2.0,"hola",3.0,True,'otro']
```

```
milista.append("otro")
```

```
milista  
[2.0, "hola", 3.0, True, 'otro']
```

```
milista[:2]
```

```
milista.append("otro")
```

```
milista  
[2.0, "hola", 3.0, True, 'otro']
```

```
milista[:2]  
[2.0, "hola"]
```

```
milista.append("otro")
```

```
milista  
[2.0,"hola",3.0,True,'otro']
```

```
milista[:2]  
[2.0,"hola"]
```

```
milista[1:]
```

```
milista.append("otro")
```

```
milista  
[2.0,"hola",3.0,True,'otro']
```

```
milista[:2]  
[2.0,"hola"]
```

```
milista[1:]  
[2.0,"hola",3.0,True]
```

```
milista.append("otro")
```

```
milista  
[2.0,"hola",3.0,True,'otro']
```

```
milista[:2]  
[2.0,"hola"]
```

```
milista[1:]  
[2.0,"hola",3.0,True]
```

```
lista2=[]
```



```
lista2  
[]
```

```
lista2  
[]
```

```
lista2  
[]
```

```
lista2.insert(1,"a")
```

```
lista2  
[]
```

```
lista2.insert(1,"a")
```

```
lista2
```

```
lista2  
[]
```

```
lista2.insert(1,"a")
```

```
lista2  
['a']
```

```
lista2  
[]
```

```
lista2.insert(1,"a")
```

```
lista2  
['a']
```

```
lista2.insert(2,"b")
```

```
lista2  
[]
```

```
lista2.insert(1,"a")
```

```
lista2  
['a']
```

```
lista2.insert(2,"b")
```

```
lista2
```

```
lista2  
[]
```

```
lista2.insert(1,"a")
```

```
lista2  
['a']
```

```
lista2.insert(2,"b")
```

```
lista2  
['a','b']
```


Tuplas

Una tupla, es como una lista: un contenedor que puede contener un número arbitrario de objetos heterogéneos. Sin embargo, las tuplas son **inmutables**, es decir, sus elementos no pueden ser modificados, por lo que no se pueden agregar y/o eliminar elementos de la tupla.

Las tuplas se pueden crear mediante la especificación de un conjunto de objetos separados por comas:

```
mi_tupla = 'hola', 4, 27.89
```

Los parentésis son opcionales cuando se crea una tupla, pero son necesarios cuando se declara una tupla vacía o la creación de tuplas anidadas:

```
mi_tupla = ()
```

Ejercicios con tuplas

```
lt = ( 1,2,True, "python" )
```

Ejercicios con tuplas

```
lt = ( 1,2,True, "python" )
```

```
lt
```

Ejercicios con tuplas

```
lt = ( 1,2,True, "python" )
```

```
lt  
(1,2, True, 'python' )
```

Ejercicios con tuplas

```
lt = ( 1,2,True, "python" )
```

```
lt  
(1,2, True, 'python' )
```

```
lt[0]=3
```

Ejercicios con tuplas

```
lt = ( 1,2,True, "python" )
```

```
lt  
(1,2, True, 'python' )
```

```
lt[0]=3  
Ups, hay un error!
```

Ejercicios con tuplas

```
lt = ( 1,2,True, "python" )
```

```
lt  
(1,2, True, 'python' )
```

```
lt[0]=3  
Ups, hay un error!
```

```
3 in lt
```

Ejercicios con tuplas

```
lt = ( 1,2,True, "python" )
```

```
lt  
(1,2, True, 'python' )
```

```
lt[0]=3  
Ups, hay un error!
```

```
3 in lt  
False
```


Los Diccionarios en Python son como los arrays asociativos de cualquier otro lenguaje (Hashmaps en Java). Se diferencian de las listas ya que los diccionarios son indizados por claves. Las claves solo podrán ser de algún tipo de dato inmutable como los enteros, los strings, las tuplas, etc.

En otras palabras un diccionario es un conjunto de pares clave-valor donde la clave es inmutable y el valor es cualquier cosa.

Uso de diccionarios

```
>>> d={}
>>> type (d)
```

Uso de diccionarios

```
>>> d={}
>>> type (d)
type 'dict'>
```

Uso de diccionarios

```
>>> d={}
>>> type (d)
type 'dict'>
```

```
>>>d={"Nombre":"Arturo Elias","Apellido":"Ant
>>> type(d)
type 'dict'>
```

Uso de diccionarios

```
>>> d={}
>>> type (d)
type 'dict'>
```

```
>>>d={"Nombre":"Arturo Elias","Apellido":"Ant
>>> type(d)
type 'dict'>
```

```
>>>d["Nombre"]
'Arturo Elias'>
```

Uso de diccionarios

```
>>> d={}
>>> type (d)
type 'dict'>
```

```
>>>d={"Nombre":"Arturo Elias","Apellido":"Ant
>>> type(d)
type 'dict'>
```

```
>>>d["Nombre"]
'Arturo Elias'>
```

```
>>>d.values()
['Arturo Elias', 'Anton']>
```

Contenido

- 1 ¿Qué necesitamos para trabajar el curso?
- 2 Introducción
 - Tipado dinámico
 - Fuertemente tipado
 - Programación orientada a objetos
 - Complementos para Python
- 3 Iniciando con Python
- 4 Operadores relacionales
- 5 Variables
- 6 Listas, Tuplas y Diccionarios
 - Listas
 - Tuplas
 - Diccionarios
- 7 Función Range()
- 8 Funciones intrínsecas

Función range

La función `range()` crea una lista de números enteros en sucesión aritmética. La función `range()` puede tener uno, dos o tres argumentos numéricos.

La función con un único argumento se escribe `range(n)` y crea una lista creciente de n términos enteros que empieza en 0 y acaba en $n - 1$ (el incremento es unitario)

```
range(8)
```


Función range

La función `range()` crea una lista de números enteros en sucesión aritmética. La función `range()` puede tener uno, dos o tres argumentos numéricos.

La función con un único argumento se escribe `range(n)` y crea una lista creciente de n términos enteros que empieza en 0 y acaba en $n - 1$ (el incremento es unitario)

```
range(8)  
[0,1,2,3,4,5,6,7]
```

Función range

La función `range()` crea una lista de números enteros en sucesión aritmética. La función `range()` puede tener uno, dos o tres argumentos numéricos.

La función con un único argumento se escribe `range(n)` y crea una lista creciente de n términos enteros que empieza en 0 y acaba en $n - 1$ (el incremento es unitario)

```
range(8)  
[0,1,2,3,4,5,6,7]
```

```
range(3,7)
```

Función range

La función `range()` crea una lista de números enteros en sucesión aritmética. La función `range()` puede tener uno, dos o tres argumentos numéricos.

La función con un único argumento se escribe `range(n)` y crea una lista creciente de n términos enteros que empieza en 0 y acaba en $n - 1$ (el incremento es unitario)

```
range(8)  
[0,1,2,3,4,5,6,7]
```

```
range(3,7)  
[3,4,5,6]
```

Función range

La función `range()` crea una lista de números enteros en sucesión aritmética. La función `range()` puede tener uno, dos o tres argumentos numéricos.

La función con un único argumento se escribe `range(n)` y crea una lista creciente de n términos enteros que empieza en 0 y acaba en $n - 1$ (el incremento es unitario)

```
range(8)  
[0,1,2,3,4,5,6,7]
```

```
range(3,7)  
[3,4,5,6]
```

```
range(4,10,2)
```

Función range

La función `range()` crea una lista de números enteros en sucesión aritmética. La función `range()` puede tener uno, dos o tres argumentos numéricos.

La función con un único argumento se escribe `range(n)` y crea una lista creciente de n términos enteros que empieza en 0 y acaba en $n - 1$ (el incremento es unitario)

```
range(8)  
[0,1,2,3,4,5,6,7]
```

```
range(3,7)  
[3,4,5,6]
```

```
range(4,10,2)  
[4,6,8]
```

Contenido

- 1 ¿Qué necesitamos para trabajar el curso?
- 2 Introducción
 - Tipado dinámico
 - Fuertemente tipado
 - Programación orientada a objetos
 - Complementos para Python
- 3 Iniciando con Python
- 4 Operadores relacionales
- 5 Variables
- 6 Listas, Tuplas y Diccionarios
 - Listas
 - Tuplas
 - Diccionarios
- 7 Función Range()
- 8 Funciones intrínsecas

Cada lenguaje de programación cuenta con un conjunto de funciones definidas que nos da soporte para ampliar los cálculos, a estas funciones, se les llama funciones intrínsecas, de tal manera que el número de ellas, es reducido, veremos que en Python, se pueden agregar más librerías especializadas que nos van a ahorrar el trabajo.

Funciones intrínsecas

```
x = -5
```


Funciones intrínsecas

```
x = -5
```

```
y = 4
```

Funciones intrínsecas

```
x = -5
```

```
y = 4
```

```
p = 3.1416
```

Funciones intrínsecas

```
x = -5
```

```
y = 4
```

```
p = 3.1416
```

```
z = '6.3'
```

Funciones intrínsecas

```
x = -5
```

```
y = 4
```

```
p = 3.1416
```

```
z = '6.3'
```

```
print int(p)
```

Funciones intrínsecas

```
x = -5
```

```
y = 4
```

```
p = 3.1416
```

```
z = '6.3'
```

```
print int(p)
```

Funciones intrínsecas

```
x = -5
```

```
y = 4
```

```
p = 3.1416
```

```
z = '6.3'
```

```
print int(p)  
3
```

Funciones intrínsecas

```
x = -5
```

```
y = 4
```

```
p = 3.1416
```

```
z = '6.3'
```

```
print int(p)  
3
```

```
abs(x)
```

Funciones intrínsecas

```
x = -5
```

```
y = 4
```

```
p = 3.1416
```

```
z = '6.3'
```

```
print int(p)
```

```
3
```

```
abs(x)
```

```
5
```



```
print float(z)
```

```
print float(z)
```

```
print float(z)  
6.0
```

```
print float(z)  
6.0
```

```
complex(x)
```

```
print float(z)  
6.0
```

```
complex(x)  
(-5+0j)
```

```
print float(z)  
6.0
```

```
complex(x)  
(-5+0j)
```

```
complex(x,y)
```

```
print float(z)  
6.0
```

```
complex(x)  
(-5+0j)
```

```
complex(x,y)  
(-5+4j)
```

```
print float(z)  
6.0
```

```
complex(x)  
(-5+0j)
```

```
complex(x,y)  
(-5+4j)
```

```
print round(p,2)
```



```
print float(z)  
6.0
```

```
complex(x)  
(-5+0j)
```

```
complex(x,y)  
(-5+4j)
```

```
print round(p,2)  
3.14
```

```
print float(z)  
6.0
```

```
complex(x)  
(-5+0j)
```

```
complex(x,y)  
(-5+4j)
```

```
print round(p,2)  
3.14
```

```
cmp(x,y)
```

```
print float(z)  
6.0
```

```
complex(x)  
(-5+0j)
```

```
complex(x,y)  
(-5+4j)
```

```
print round(p,2)  
3.14
```

```
cmp(x,y)  
-1
```

Operación	Descripción
<code>int(x)</code>	Convierte x a entero
<code>long(x)</code>	Convierte x a entero largo
<code>float(x)</code>	Convierte x a punto flotante
<code>complex(x)</code>	Convierte x al complejo $x+0j$
<code>complex(x,y)</code>	Convierte al complejo $x+yj$

Función	Descripción
<code>abs(x)</code>	Valor absoluto de x
<code>max(sucesion)</code>	Mayor elemento de la sucesión
<code>min(sucesion)</code>	Menor elemento de la sucesión
<code>round(x,n)</code>	Redondea x al decimal n
<code>cmp(x,y)</code>	Devuelve -1 , 0 , 1 si $x < y$, $x == y$, $x > y$