CURSO INICIACIÓN PYTHON

MODULO I - ESTRUCTURA Y ELEMENTOS DEL LENGUAJE

ROBOTechnics
Robótica Educativa

Para el primer día de clase es necesario traer instalado:

Python, lo puedes descargar desde este enlace
https://www.python.org/downloads/
tienes que elegir la versión 3.4 o superior.

MARCAR LA OPCIÓN DE INCLUIR PYTHON 3.5 EN EL PATH

LiClipse, el cual se puede descargar desde aquí
http://www.liclipse.com/download.html

Instalación de Python en Windows8:
http://recursospython.com/guias-y-manuales/instalar-python-3-en-windows-8/

TEMARIO

MODULO I - ESTRUCTURA Y ELEMENTOS DEL LENGUAJE

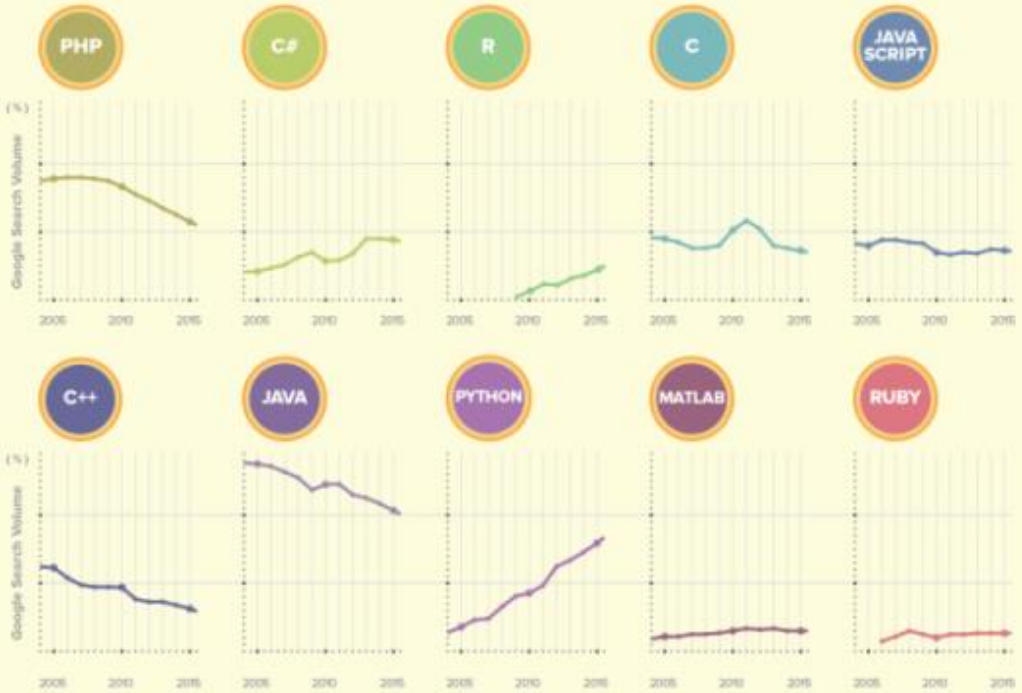
- Introducción a Python
- Tipos de datos
- Operadores Aritméticos
- Estructuras de Control de Flujo

Introducción a Python

CURSO DE PYTHON

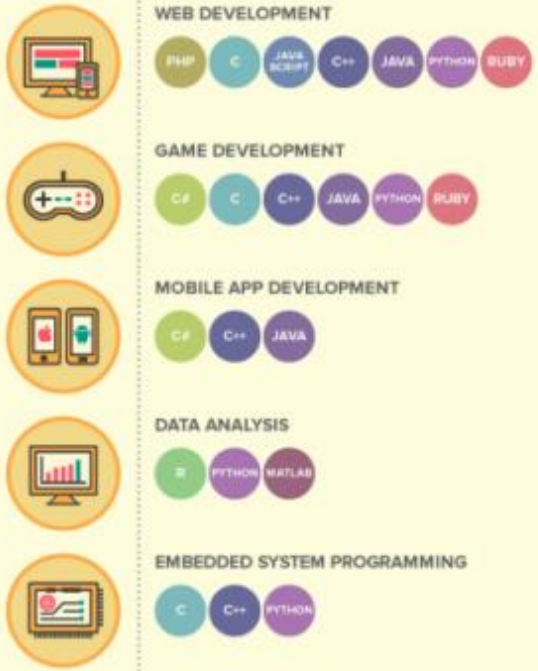
POPULARITY

Python, used at both Google and Facebook, is the fastest growing language of the bunch based on the percentage of Google searches in the US for Python tutorials versus other language tutorials. Java, PHP, and C++ held a lot of interest a decade ago, but searches for related tutorials been on the decline (though there are still a ton of Java job openings across the country, including at Apple!).



APPLICATION

If career flexibility is important to you, learning Python or C++ allows you to work in most major types of programming, from creating games to building embedded systems. If you choose JavaScript or PHP, however, be prepared for a career in web development. Likewise, studying Matlab or R primarily qualifies you for a career in data analysis.



SALARY

If money drives you, study Ruby, Matlab, or Python. They have the three highest average salaries of the top ten languages and are the only languages that pay over $100,000 per year on average. If you study PHP or C#, expect a lower (though still lucrative!) salary — both average a little below $90,000 per year.

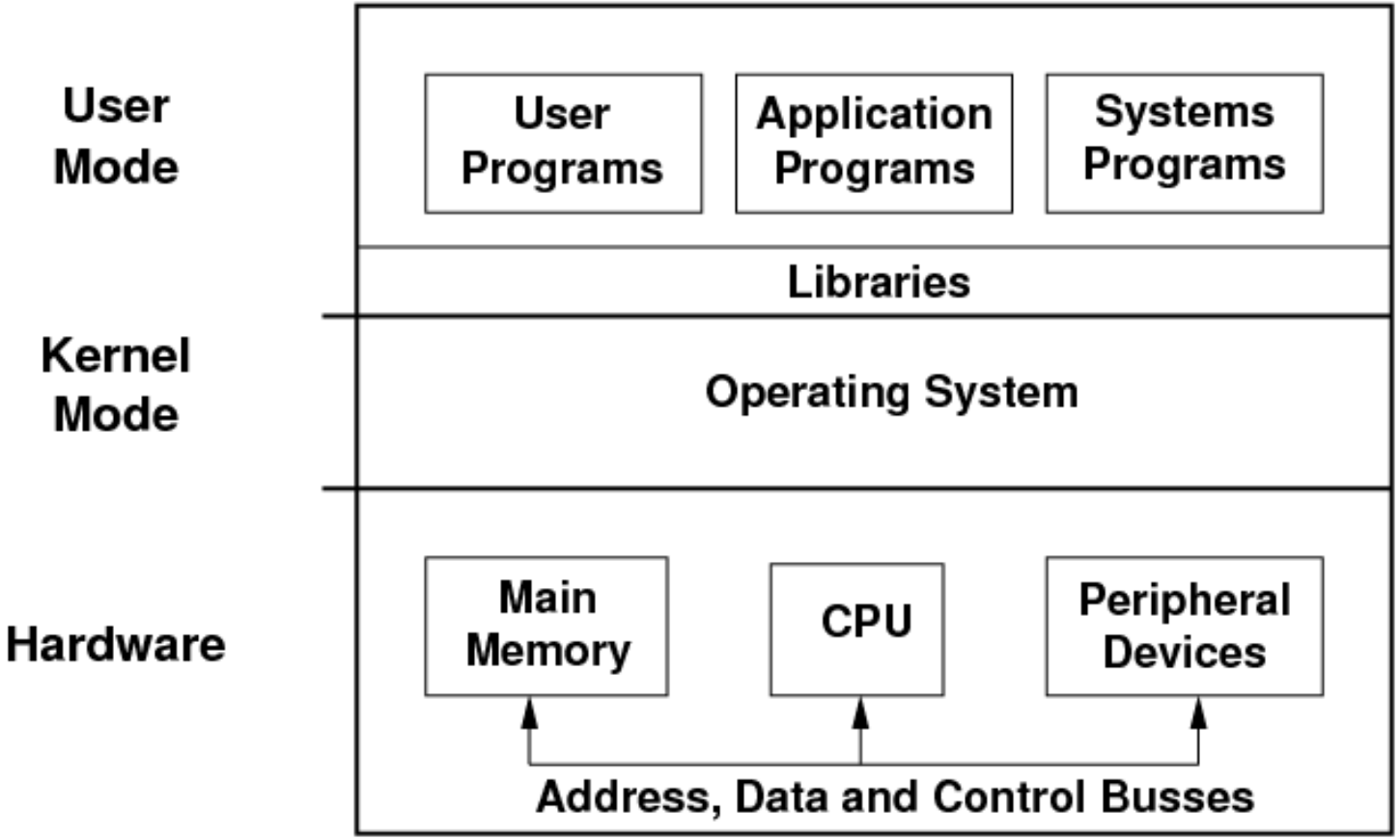
Introducción a Python

Se llama **Programación** a la implementación de un algoritmo en un determinado lenguaje de programación, para resolver un problema

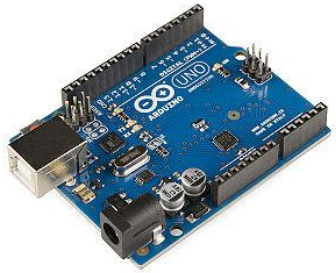
Algoritmo es una secuencia no ambigua, finita y ordenada de instrucciones que han de seguirse para resolver un problema

Programa (Software en inglés) es una secuencia de instrucciones que una computadora puede interpretar y ejecutar

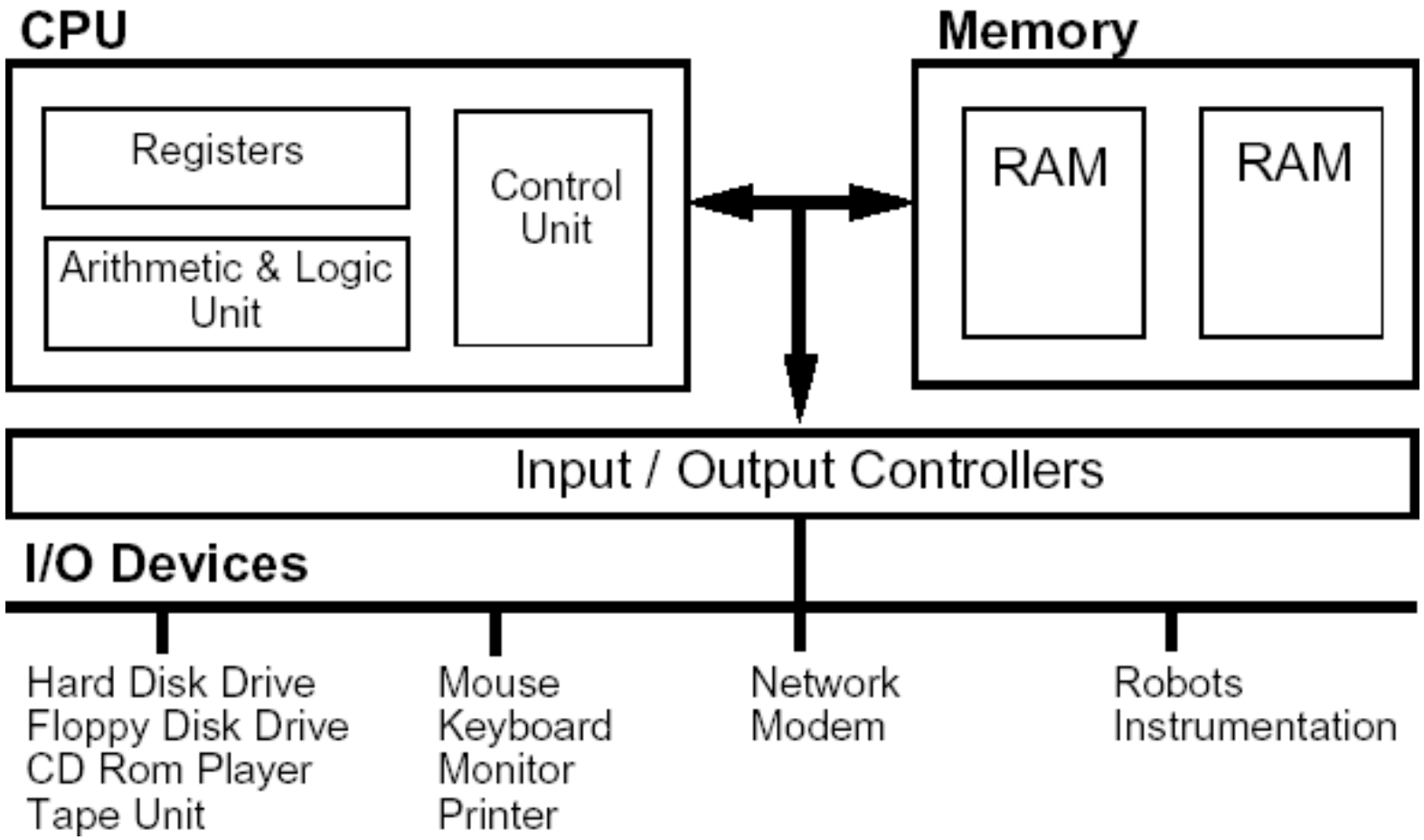
ARQUITECTURA DE ORDENADORES



Linux



ARQUITECTURA DE ORDENADORES



Introducción a Python

- Niveles de lenguajes de programación

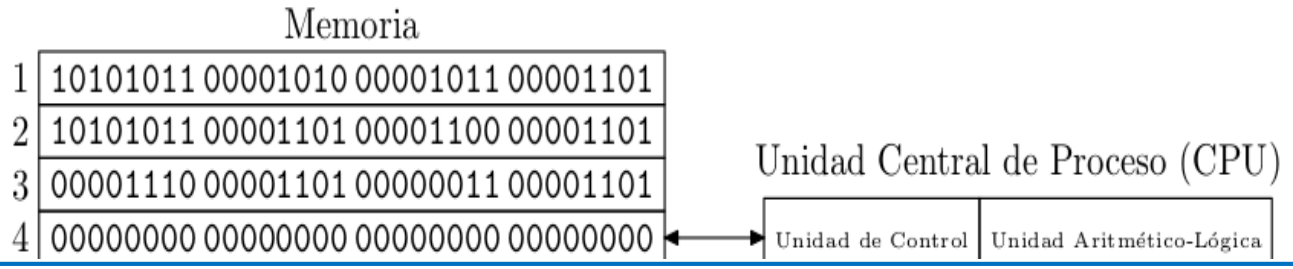
Binario: es el lenguaje que puede ejecutar directamente la CPU mediante 0 y 1.

Cada familia de ordenadores dispone de su propio repertorio de instrucciones, es decir, de su propio código máquina.

Un programa que, por ejemplo, calcula la media de tres números almacenados en las posiciones de memoria 10, 11 y 12, respectivamente, y deja el resultado en la posición de memoria 13, podría tener el siguiente aspecto expresado de forma comprensible para nosotros:

Dividir contenido de dirección 13 por 3 y dejar resultado en dirección 13
Detener

En realidad, el contenido de cada dirección estaría codificado como una serie de unos y ceros, que el aspecto real de un programa como el descrito arriba podría ser éste:



Introducción a Python

- Niveles de lenguajes de programación

Ensamblador: Empleado en videojuegos para programar la tarjeta gráfica y conseguir aplicaciones en tiempo real

```
SUM #13, #12, #13
DIV #13, 3, #13
FIN
```

Lenguajes de alto nivel: Pascal, Cobol, Python

Lenguajes orientados a objetos: C++, Java, Python

Introducción a Python

Compilación: Traduce las instrucciones de alto nivel al código binario que entiende el ordenador.

Realizado por compiladores.

Recoge un fichero en texto y genera otro fichero en lenguaje binario.

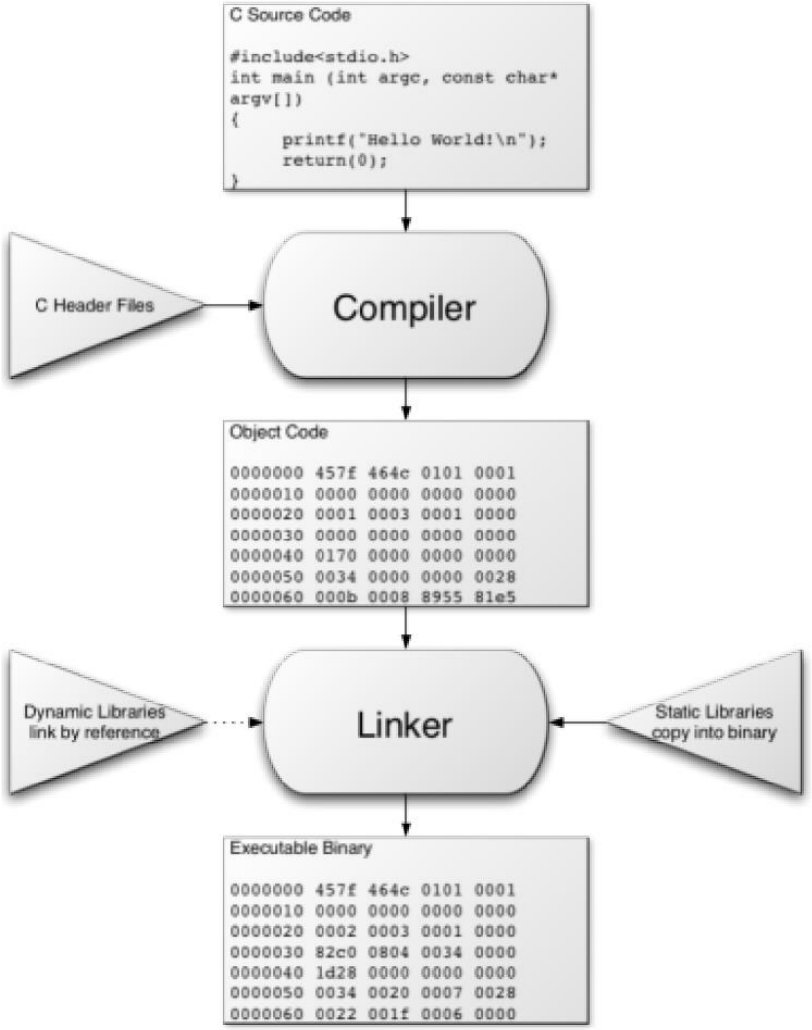
Desventajas de la compilación:

Se genera un fichero binario para un ordenador y Sistema Operativo concretos.

De ahí que en las descargas existan opciones de descarga para Windows 10, Linux, IOS

Ventajas:

Su ejecución es mucho más rápida



Esto implica que los programas deben ser compilados antes de poder ser ejecutados. En algunos lenguajes esta compilación se hace con todo el código mientras que en otros se va compilando línea por línea (lenguajes interpretados). Los lenguajes interpretados son por lo general mas lentos para ejecutarse que los programas compilados en su totalidad. Un problema asociado a los lenguajes compilados es que para cada cambio que se quiera probar hay que recompilar todo el código.

Una solución intermedia es hacer una compilación previa contra una máquina virtual, este es el caso de
Python (junto con Java, C#, VB.net y otros).
Las principales consecuencias de la compilación son:

- Tiempo de compilación
- Aceleración en la ejecución del software
- Software dependiente de una plataforma

Introducción a Python

- **En el caso de Java**

  Se genera un ByteCode que es una capa intermedia de SW que es independiente de la plataforma

  A continuación se emplea una máquina virtual que se adapta ya a la plataforma de hardware y OS empleado en cada ordenador.

Introducción a Python

- **En el caso de Python**:

  Al iniciar, se arranca una máquina virtual que enlaza el programa Python y el sistema operativo

  Al iniciar un programa, se compila tanto el módulo principal como los accesorios.

  Se genera un archivo .pyc que evita volver a compilar si el programa no ha cambiado, ahorrando así tiempo en el arranque.

  Este .pyc contiene el código explotable de la máquina virtual y es independiente de la plataforma.

  Antes de ejecutarse, se comprueba que el código es correcto. Ejemplo de comprobación del código. En este caso, obtendremos un error

```
'''
Operaciones con boolean
''''
SyntaxError: EOL while scanning string literal
```

Python

- Es un lenguaje de programación interpretado y multiplataforma
- Lenguaje de Alto Nivel: Al programar en Python no nos debemos preocupar por detalles de bajo nivel, (como manejar la memoria empleada por el programa).

- Es Incrustable. Se puede insertar lenguaje Python dentro un programa C/C++ y de esta manera ofrecer las facilidades del scripting.

- Es lenguaje de programación multiparadigma, ya que soporta:
  - orientación a objetos
  - programación imperativa (pueden emplear etiquetas por líneas como COBOL, BASIC etc)
  - y, en menor medida, programación funcional
  - Permite la programación concurrente, es decir, realización de tareas de forma simultánea.

Python

Es un lenguaje con identacion

- Por ejemplo, en C

```
int factorial(int x)
{
    if (x < 0 || x % 1 != 0){
        printf("x debe ser un numero entero mayor o igual a 0");
        return -1; //Error
    }
    else if (x == 0)
        return 1;
    else
        return x * factorial(x - 1);
}
```

- En Python

```
def factorial(x):
    assert x >= 0 and x % 1 == 0, "x debe ser un entero mayor o igual a 0."
    if x == 0:
        return 1
    else:
        return x * factorial(x - 1)
```

Python

Es un lenguaje Open Source, del que existen diversas implementaciones del lenguaje:

- CPython es la implementación original (realizada en C), disponible para varias plataformas en el sitio oficial de Python.
- IronPython es la implementación para .NET
- Jython es la implementación hecha en Java
  - Todo lo que puedes hacer con Java también lo puedes hacer con Python
  - Incluso puedes acceder a través de Python a las API de Java si usas Jython (http://www.jython.org)

Cada una de estas versiones aprovechan las ventajas del programa en el que están diseñadas

Introducción a Python

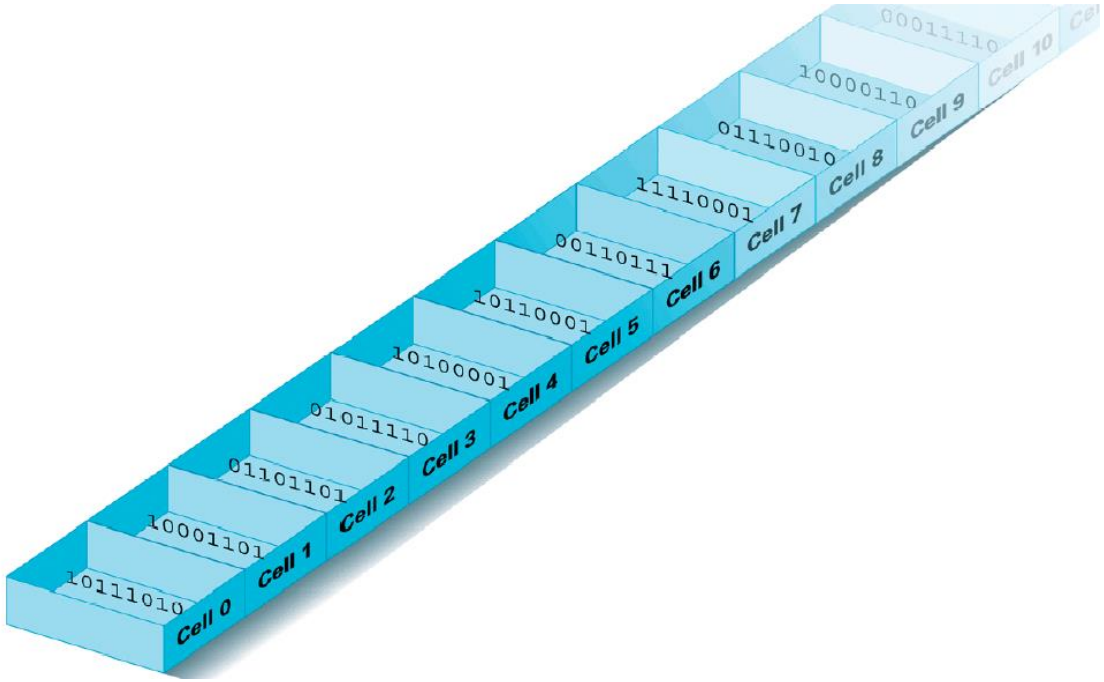
Python tiene tipado dinámico de las variables:

En Java, por ejemplo, escribiríamos:

```
String c = "Hola Mundo";
int e = 23;
```

Sin embargo, en Python, los tipos de variables pueden cambiar de forma dinamica:

```
c = “Hola Mundo”
C= 23
```



La variables son objetos que apuntan a una dirección de memoria. Cuando se cambian de forma dinámica, el mismo objeto apunta a otra dirección y la dirección antigüa se borra automáticamente

Introducción a Python: Paradigmas de programación

Existen decenas de paradigmas de programación, los más importantes son:

- **Procedural**: En la programación procedural se le dan ordenes directas a la computadora, que van cambiando el estado del programa. Ejemplo: BASIC

- **Estructurada**: Es un caso particular de programación procedural, la diferencia es que hay un flujo ordenado (jerárquico) con alguna combinación de los siguientes elementos: Secuencias, selección y repetición. Se reutilizan bloques de código y es relativamente fácil seguir la lógica del programa. Ejemplo: C, Pascal.

- **Orientada a objetos**: Se usan estructuras de datos que tienen datos y métodos (objetos) con sus interacciones para diseñar programas. Ejemplos: Java, C++

Python suele ser clasificado como "multiparadigma" debido a que puede ser usado tanto de manera estructurada como orientado a objetos. Esto depende de las preferencias del programador, no impone uno de los dos métodos

Diferencias entre Python 2.7 y 3.0

- Print es una función en python 3
  - En Python 2 → print "hola mundo"
  - En Python 3 → print ("hola mundo")

- División de números enteros. print (3 / 2)
  - En Python 2 → = 1
  - En Python 3 → = 1.5

- Las “cadenas” (“strings”) son Unicode de forma predeterminada en python 3

- Input es una cadena de texto en python 3

- La función next() y el método .next()
  - En python 2 se puede usar next tanto como una función next(algo) o como un método algo.next(), sin embargo en Python 3 solo se puede usar como función:

Diferencias entre Python 2.7 y 3.0

- Manejo de Exepciones
  - En python 2 se acepta las dos maneras de escribir una excepción (sin paréntesis o con paréntesis como si fuera una función), en cambio en python 3 solo se acepta de la segunda forma.
- Hay mas cambios … pero los dejamos de momento

Introducción a Python

- En windows: abrir ventana de terminal y poner

  `python mifichero.py` (En Linux `python3 mifichero.py`)

  Si no se ejecuta, es necesario comprobar las variables de entorno que es donde se buscan los programas que se pueden ejecutar.

  En Windows, en “Mi Equipo” ir a Propiedades / Config Avanzada del sistema/Variables de entorno/Variables del Sistema

  En la variable PATH hay que añadir el directorio donde esté el intérprete de Python
  Normalmente:
  Se añade con “;” + directorio y se cierra el terminal para que se actualice

Hemos invocado al intérprete para que coja el código fuente y lo ejecute
Previamente lo ha analizado para ver que cumple las reglas del lenguaje
Si se genera el código intérprete aparecerá el fichero: `mifichero.pyc`

Tipos de Datos

TIPOS DE DATOS

Object type	Example literals/creation
Numbers	`1234, 3.1415, 999L, 3+4j, Decimal`
Strings	`'spam', "guido's"`
Lists	`[1, [2, 'three'], 4]`
Dictionaries	`{'food': 'spam', 'taste': 'yum'}`
Tuples	`(1,'spam', 4, 'U')`
Files	`myfile = open('eggs', 'r')`
Other types	Sets, types, None, Booleans

PALABRAS RESERVADAS EN PYTHON

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

ESTRUCTURA Y ELEMENTOS DEL LENGUAJE

Secuencia de Escape	Significado
\newline	Backslash and newline ignored
\\	Backslash (\)
\'	Single quote (')
\"	Double quote (")
\a	ASCII Bell (BEL)
\b	ASCII Backspace (BS)
\f	ASCII Formfeed (FF)
\n	ASCII Linefeed (LF)
\r	ASCII Carriage Return (CR)
\t	ASCII Horizontal Tab (TAB)
\v	ASCII Vertical Tab (VT)
\ooo	Character with octal value ooo
\xhh	Character with hex value hh

Introducción a Python

- Ejercicio: Generar fichero de texto plano con nombre: mifichero.py

  Contenido del programa

```
# esto es una cadena
c = "Hola Mundo"

d = 'Hola Mundo'
# y esto es un entero
e = 23
# podemos comprobarlo con la función type
print (type(c))
print (type(e))
'''
Esto en un comentario multipl2
'''
input()
```

  OJO con la identación: No se pueden dejar espacios, tabulaciones etc a la derecha. No poner caracteres especiales ni siguiera en las líneas de comentarios

TIPOS DE DATOS: Números

- Enteros: int y long
- Reales (decimales): float

Operaciones:

Operador	Descripción	Ejemplo
+	Suma	`r = 3 + 2  # r es 5`
-	Resta	`r = 4 - 7  # r es -3`
-	Negación	`r = -7  # r es -7`
*	Multiplicación	`r = 2 * 6  # r es 12`
**	Exponente	`r = 2 ** 6  # r es 64`
/	División	`r = 3.5 / 2  # r es 1.75`
//	División entera	`r = 3.5 // 2 # r es 1.0`
%	Módulo	`r = 7 % 2  # r es 1`

TIPOS DE DATOS: Cadenas de texto

La codificación ASCII se definió en 1960 y se desarrolló para tecnologías de 8 bits.
ASCII estándar, concretamente, utiliza los 7 primeros bits para codificar la información y el número 8 es el bit de paridad usado para controlar errores.
Con los 7 bits se diseñó un juego de 128 caracteres (en binario 2 elevado a 7), partiendo del carácter identificado con el número 0 y terminando en el número 127.

(espacio) ! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\ ] ^ _
` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~
Está codificación estaba pensada para el alfabeto inglés

TIPOS DE DATOS: Cadenas de texto

Caracteres de control ASCII

DEC	HEX	Simbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de linea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

Caracteres ASCII imprimibles

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

elCodigoASCII.com.ar

ASCII extendido

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
128	80h	Ç	160	A0h	á	192	C0h	└	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	┴	225	E1h	ß
130	82h	é	162	A2h	ó	194	C2h	┬	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	├	227	E3h	Ò
132	84h	ä	164	A4h	ñ	196	C4h	─	228	E4h	õ
133	85h	à	165	A5h	Ñ	197	C5h	┼	229	E5h	Õ
134	86h	å	166	A6h	ª	198	C6h	ã	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	Ã	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	╚	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	╔	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	╩	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	╦	235	EBh	Ù
140	8Ch	î	172	ACh	¼	204	CCh	╠	236	ECh	ý
141	8Dh	ì	173	ADh	¡	205	CDh	═	237	EDh	Ý
142	8Eh	Ä	174	AEh	«	206	CEh	╬	238	EEh	¯
143	8Fh	Å	175	AFh	»	207	CFh	¤	239	EFh	´
144	90h	É	176	B0h	░	208	D0h	ð	240	F0h	
145	91h	æ	177	B1h	▒	209	D1h	Ð	241	F1h	±
146	92h	Æ	178	B2h	▓	210	D2h	Ê	242	F2h	‗
147	93h	ô	179	B3h	│	211	D3h	Ë	243	F3h	¾
148	94h	ò	180	B4h	┤	212	D4h	È	244	F4h	¶
149	95h	ò	181	B5h	Á	213	D5h	ı	245	F5h	§
150	96h	û	182	B6h	Â	214	D6h	Í	246	F6h	÷
151	97h	ù	183	B7h	À	215	D7h	Î	247	F7h	¸
152	98h	ÿ	184	B8h	©	216	D8h	Ï	248	F8h	°
153	99h	Ö	185	B9h	╣	217	D9h	┘	249	F9h	¨
154	9Ah	Ü	186	BAh	║	218	DAh	┌	250	FAh	·
155	9Bh	ø	187	BBh	╗	219	DBh	█	251	FBh	¹
156	9Ch	£	188	BCh	╝	220	DCh	▄	252	FCh	³
157	9Dh	Ø	189	BDh	¢	221	DDh	¦	253	FDh	²
158	9Eh	×	190	BEh	¥	222	DEh	Ì	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	▀	255	FFh	

TIPOS DE DATOS: Cadenas de texto

A lo largo del tiempo surgieron otras codificaciones hasta que la globalización y la necesidad de intercambio de información en distintos sistemas y en diferentes idiomas hizo concentrar esfuerzos en el desarrollo de un proyecto universal de codificación llamado Unicode.

"Unicode es un estándar de codificación de caracteres diseñado para facilitar el tratamiento de textos de múltiples lenguajes, incluido los basados en ideogramas o aquellos usados en textos de lenguas muertas. El término Unicode proviene de los objetivos perseguidos durante el desarrollo del proyecto: universalidad, uniformidad y unicidad".

En Unicode los caracteres alfabéticos, los ideogramas y los símbolos se tratan de forma equivalente y se pueden mezclar entre sí en un mismo texto, es decir, es posible representar en un mismo párrafo caracteres del alfabeto árabe, cirílico, latino, ideogramas japoneses y símbolos musicales.
Para hacernos una idea del volumen de caracteres que es capaz de representar Unicode, señalar que su versión 5.1 contiene más de 100.000.

TIPOS DE DATOS: Cadenas de texto. MÉTODOS DISPONIBLES

```
>>> print('ACME', 50, 91.5)
ACME 50 91.5
>>> print('ACME', 50, 91.5, sep=',')
ACME,50,91.5
>>> print('ACME', 50, 91.5, sep=',', end='!!\n')
ACME,50,91.5!!
>>>
```

TIPOS DE DATOS: Cadenas de texto

Concatenación

```
a = "uno"
b = "dos"
c = a + b # c es "unodos"
print(c)
c = a * 3 # c es "unounouno"
print(c)

''' Operaciones con variables
'''
a=14
b=12
c = a+b
print(a, "hola",  b, '=', c)
```

TIPOS DE DATOS: Cadenas de texto

Concatenación

```
'''
''' Operaciones con variables y texto
'''
a=14
b=12
c = a+b
print(a+b)
print(a, "hola",  b, '=', c)
print(a+ "hola"+  b+ '='+ c) → ERROR
```

A la hora de concatenar número y cadenas de texto, es necesario convertir números as variables de texto:

```
print(str(a)+ "hola"+ str(b)+ '='+ str(c))
```

TIPOS DE DATOS: Boolean

True, False → OJO → primera letra siempre en mayúsculas

– Importantes en expresiones

– Realmente representan 0 y 1

- Para convertir a tipo Boolean:
  bool("True")

Operador	Descripción	Ejemplo
and	¿se cumple a y b?	r = True and False # r es False
or	¿se cumple a o b?	r = True or False # r es True
not	No a	r = not True # r es False

```
'''
Operaciones con boolean
'''
a=1
b=0
print ("a=", a, " b= ", b)
print ("a and b -->", a and b)
print ("a or b  -->", a or b)
```

Operadores Aritméticos

TIPOS DE DATOS: Operadores de comparación

Normalmente se usan en combinación con los Boolean

Operador	Descripción	Ejemplo
==	¿son iguales a y b?	r = 5 == 3 # r es False
!=	¿son distintos a y b?	r = 5 != 3 # r es True
<	¿es a menor que b?	r = 5 < 3 # r es False
>	¿es a mayor que b?	r = 5 > 3 # r es True
<=	¿es a menor o igual que b?	r = 5 <= 5 # r es True
>=	¿es a mayor o igual que b?	r = 5 >= 3 # r es True

OJO → a = 5 le a
→ a == 5 igual a 5?. El resultado es un Boolean

TIPOS DE DATOS: Operadores de incremento

```
'''
Operaciones con operadores de incremento
'''
a=23
print("a=23  -->", a)
a=a+1
print("a=a+1  -->", a)
a+=1
print("a+=1  -->", a)
#a++ → ERROR
a*=2  #a=a*2
print("a*=2  -->", a)
a/=2  #a=a/2
print("a/=2  -->", a)
a-=2  #a=a-2
print("a-=1  -->", a)
a**=2 #a=a**2
print("a**=1  -->", a)
```

```
a=23  --> 23
a=a+1  --> 24
a+=1  --> 25
a*=2  --> 50
a/=2  --> 25.0
a-=1  --> 23.0
a**=1  --> 529.0
```

CONVERSIONES DE TIPO

```
a = 10
#print ("La variable a es: " + a )
print ("La variable a es: " + str(a)) # concatenamos en un string y luego
imprimimos
print ("La variable a es: ", a) # imprimimos una cadena y una variable (añade un
espacio por defecto)

b = 2.3
c = 9.7657522334
print (b, " es del tipo ", type(b))
print (c, " es del tipo ", type(c))
d= "345"

print (d+str(a)) # convertimos numero a cadena
print (int(d)+a) # convertimos cadena a numero
print (b+c)
Print (a+b)
```

PETICIÓN DE DATOS DEL USUARIO

```
'''
Peticiones de datos al usuario
'''
print ("introduce un dato")
a = input()
print(a, type(a)) # el valor introducido siempre es una cadena

e = input ("introduce un dato "+d)
print (e)
print ("introduce un dato ", input(a))


print ("El resultado es ", int(input("Valor 1: ")) + int(input("Valor 2: ")))
# probar a meter 12.4334 en la linea anterior

print ("El resultado es ", float(input("Valor 1: ")) + float(input("Valor 2: ")))
```

EL “PROBLEMA” DE LOS NÚMEROS CON COMA FLOTANTE

El resultado de las operaciones con números flotantes puede ser inesperado:

```
>>> 0.1 + 0.1 + 0.1 - 0.3
5.5511151231257827e-17
```

La causa de este problema es que Python no esconde el hecho que las operaciones que involucran numeros de punto flotante no son exactas debido a que hay un error inherente al pasar internamente los números de la base 2 (la que usa realmente el hardware) a la base 10.

Este error ocurre con todos los lenguajes, la diferencia es que la mayoría de los lenguajes oculta este hecho usando algún tipo de redondeo.

En Python este redondeo hay que hacerlo de manera:

```
>>> round(0.1 + 0.1 + 0.1 – 0.3, 1)
0.
```

EL “PROBLEMA” DE LOS NÚMEROS CON COMA FLOTANTE

Un ordenador no es capaz de resolver de forma continua todos los datos, sino que lo hace en forma de escalones. A mayor número de bits, mayor es la precisión alcanzada.

En el ejemplo anterior:

```
>>> 0.1 + 0.1 + 0.1 - 0.3
                    5.5511151231257827e-17
```

Es decir, para 64 bits, el error en la precisión es del orden de 10E-17

Debido precisamente a ese salto en escalones en los cálculos del ordenador, su sistema de medición de la información es el siguiente:

1 bit es la unidad mínima, y puede valer 0 o 1.
1 byte es un grupo de 8 bits.
1 kilobyte (a veces escrito como KB) es un grupo de 1024 bytes (es decir, 8192 bits o **2**10)**.
1 megabyte (MB) es un grupo de 1024 KB (es decir, 1048576 bytes). 1 gigabyte (GB) es un grupo de 1024 MB.

EL “PROBLEMA” DE LOS NÚMEROS CON COMA FLOTANTE

```
>>> a = 4.2
>>> b = 2.1
>>> a + b
6.300000000000001
>>> (a + b) == 6.3
False
>>>
```

```
>>> from decimal import Decimal
>>> a = Decimal('4.2')
>>> b = Decimal('2.1')
>>> a + b
Decimal('6.3')
>>> print(a + b)
6.3
>>> (a + b) == Decimal('6.3')
True
>>>
```

EL “PROBLEMA” DE LOS NÚMEROS CON COMA FLOTANTE

```
>>> from decimal import localcontext
>>> a = Decimal('1.3')
>>> b = Decimal('1.7')
>>> print(a / b)
0.7647058823529411764705882353
>>> with localcontext() as ctx:
... ctx.prec = 3
... print(a / b)
0.765
>>> with localcontext() as ctx:
... ctx.prec = 50
... print(a / b)
...
0.76470588235294117647058823529411764705882352941176
>>>
```

EL “PROBLEMA” DE LOS NÚMEROS CON COMA FLOTANTE

you want to use exponential notation, change the format to an
case you want used for the exponential specifier. For example:

```
>>> format(x, 'e')
'1.234568e+03'
>>> format(x, '0.2E')
'1.23E+03'
```

```
>>> x = 1234.56789
>>> # Two decimal places of accuracy
>>> format(x, '0.2f')
'1234.57'
>>> # Right justified in 10 chars, one-digit accuracy
>>> format(x, '>10.1f')
' 1234.6'
>>> # Left justified
>>> format(x, '<10.1f')
'1234.6 '
>>> # Centered
>>> format(x, '^10.1f')
' 1234.6 '
>>> # Inclusion of thousands separator
>>> format(x, ',')
'1,234.56789'
>>> format(x, '0,.1f')
'1,234.6'
>>>
```

OTROS FORMATOS

```
>>> x = 1234
>>> bin(x)
'0b10011010010'
>>> oct(x)
'0o2322'
>>> hex(x)
'0x4d2'
>>> format(x, 'b')
'10011010010'
>>> format(x, 'o')
'2322'
>>> format(x, 'x')
'4d2'
>>>
```

```
>>> from fractions import Fraction
>>> a = Fraction(5, 4)
>>> b = Fraction(7, 16)
>>> print(a + b)
27/16
>>> print(a * b)
35/64
>>> # Getting numerator/denominator
>>> c = a * b
>>> c.numerator
35
>>> c.denominator
64
>>> # Converting to a float
>>> float(c)
0.546875
>>> # Limiting the denominator of a value
>>> print(c.limit_denominator(8))
4/7
>>> # Converting a float to a fraction
>>> x = 3.75
>>> y = Fraction(*x.as_integer_ratio())
>>> y
Fraction(15, 4)
```

¿Alguna Pregunta?