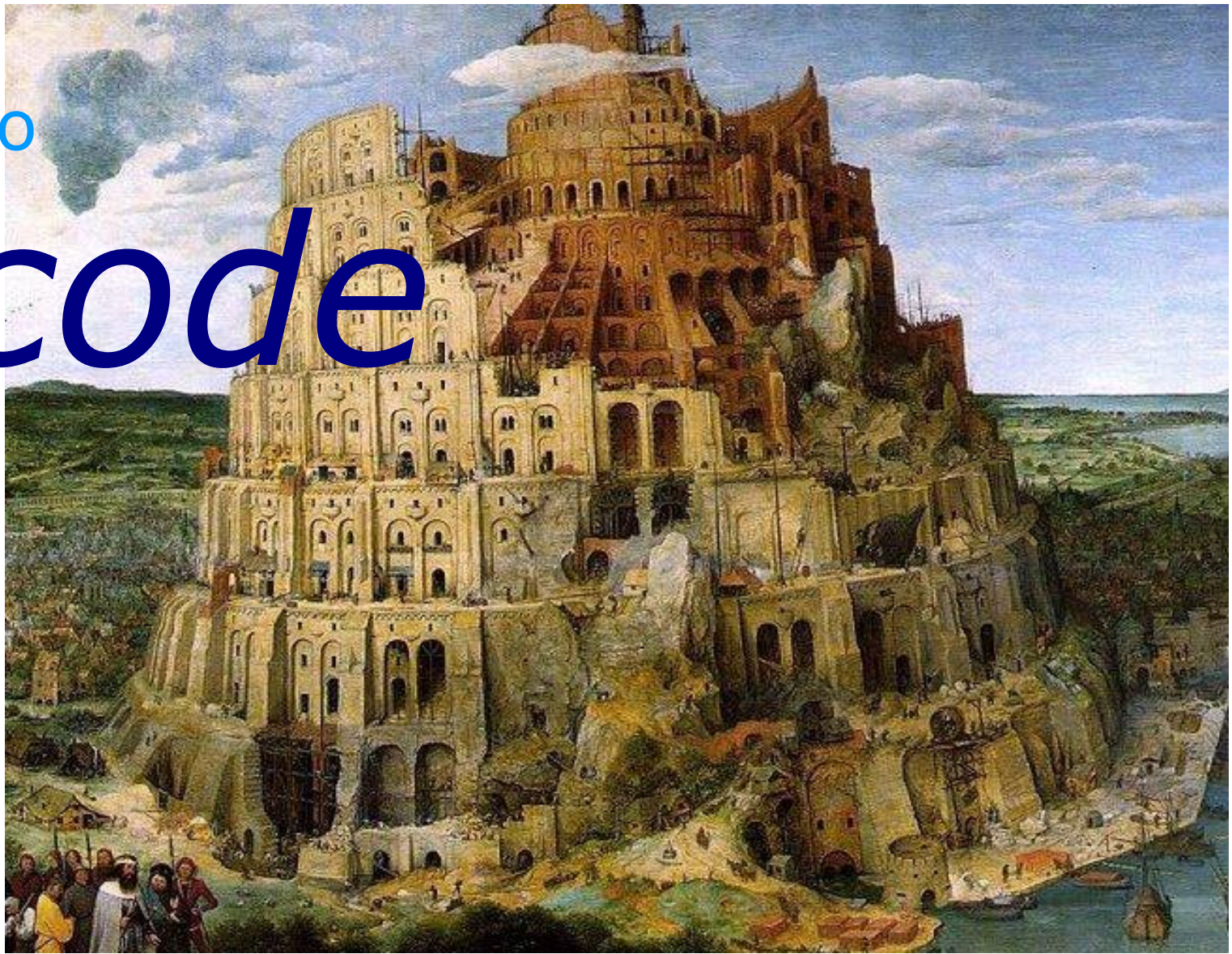


Entendiendo

Unicode



Facundo Batista

facundo@taniquetil.com.ar

<http://www.taniquetil.com.ar>



¿Qué es Unicode?

- x Un estándar
- x ¿Pero en qué consiste?
 - x Un repertorio de más de 100 mil caracteres
 - x Planillas de códigos para referencia visual
 - x Metodologías de codificación
 - x Codificaciones estándares
 - x Enumeración de propiedades de los caracteres
 - x Etc...

¿Pero esto cómo
se come?



Masticando un Code Chart

- x Una planilla de códigos es una **tabla** con el dibujo de cada caracter
- x El **mapa** completo de caracteres de Unicode está **divido** en varios **codecharts**
- x Por ejemplo:

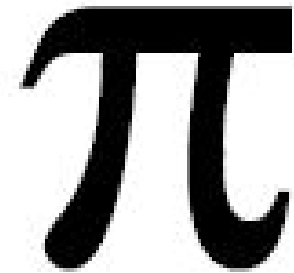
- x Basic Latin
- x Latin-1
- x Greek
- x Runic

0022	0032	0042	0052	0062	30	Π	ü	π	θ						
#	3	␣	␣	␣	30	03A0	03B0	03C0	03D0						
0023	0033	NB SP	◦	À	31	P	α	ϣ	†	‡	1	Υ	ϕ		
\$	4	0040	00B0	00C0	31	03A1	03B1	16A0	16B0	16C0	16D0	16E0	16F0		
0024	0034	¡	±	Á	32		β	ϣ	℞	ℓ	†	*			
%	5	00A1	00B1	00C1	32		03B2	16A1	16B1	16C1	16D1	16E1			
0025	0035	¢	2	Â	32	Σ	γ	Π	<	†	ℬ	ℳ			
		00A2	00B2	00C2				16A2	16B2	16C2	16D2	16E2			
		£	3	Ã	Ó	ã	ó	ℵ	℥	↵	℥	℥			
		00A3	00B3	00C3	00D3	00E3	00F3								
		ü	ü	ä	ô	ö	û								

Ver todas las planillas

Un estándar con carácter

- x **Por cada caracter** tenemos:
 - x Un glifo (de referencia, no prescriptivo)
 - x Un nombre
 - x Un número
 - x Más info
- x **Ejemplo:**
 - x 03C0
 - x GREEK SMALL LETTER PI
 - x math constant 3.141592...

A large, bold, black Greek letter Pi symbol (π) is displayed on the right side of the slide.

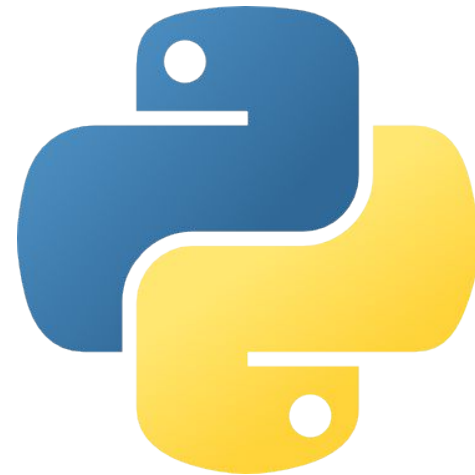
¿Y en Python?

- x Definimos el caracter según su código

```
>>> pi = u"\u03c0"  
>>> len(pi)  
1
```

- x ¿Y cómo se ve?

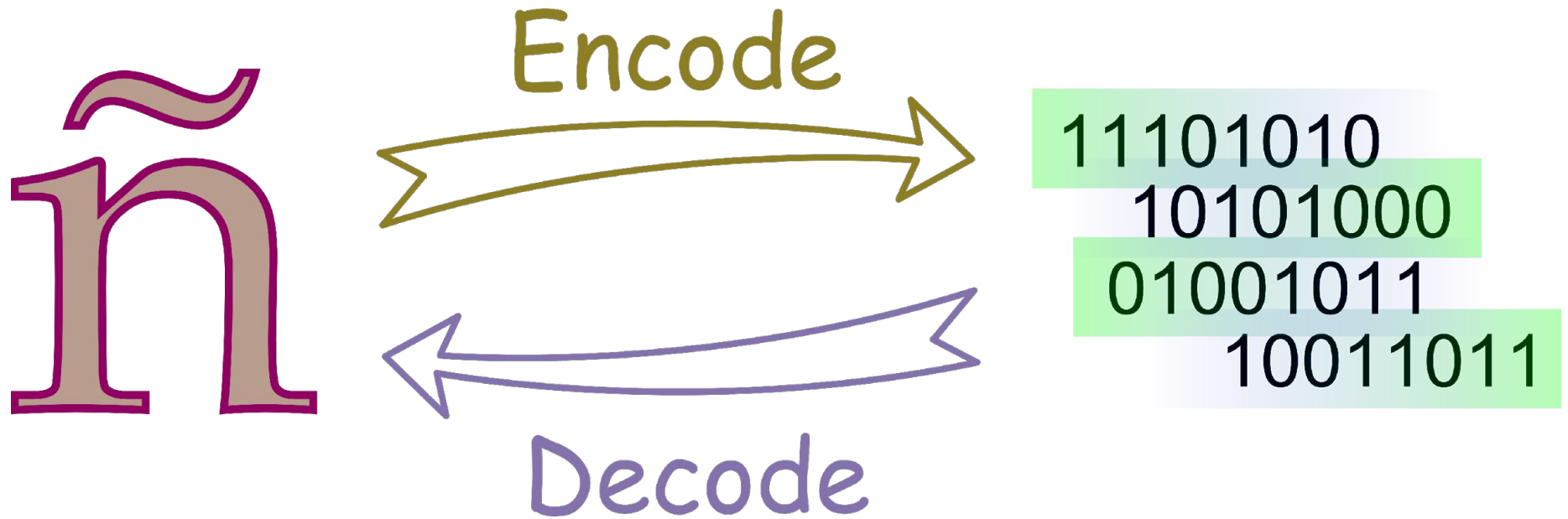
```
>>> print pi  
π  
>>> pi  
u'\u03c0'  
>>> pi == u"π"  
True
```



Bititos

- x Unicode es muy lindo... ¿pero como lo persistimos o transmitimos?
- x Tenemos que traducir los caracteres Unicode a bits.
- x Codificar o Encode
 - x Pasar de un caracter Unicode
a una secuencia de bytes
- x Decodificar o Decode
 - x Pasar de una secuencia de bytes
a un caracter Unicode

Vamos de nuevo



¿No es directo?

- x Necesitamos codificar la información de una **forma estándar**
- x No es exclusivo de Unicode, nos pasa con los números también
- x Ejemplo: Números
 - x Negativos: Complemento a 1 o a 2
-3 puede ser (bin) 1100 o 1101
 - x Punto flotante: Precisión simple y cuádruple
1.0 puede ser (hex) 3F80 0000 o
3FFF 0000 0000 0000 0000 0000 0000 0000

Codificando Unicode

- x ¿Cómo representamos en bits un caracter Unicode?
 - x Tenemos que inventar un código
 - x Definir cuantos bits usará
 - x Definir las reglas de conversión
- x Ya tenemos muchos códigos estándares
 - x ASCII
 - x Latin-1 (ISO-8859-1)
 - x UTF-8, -16, -32
 - x etc

Distintas formas de lo mismo

- x Distintos encodings, distintos bytes:

```
>>> enie = u"ñ"  
>>> enie.encode("latin1")  
'\xf1'  
>>> enie.encode("utf8")  
'\xc3\xbf'  
>>> enie.encode("utf16")  
'\xff\xfe\xf1\x00'
```

- x ¿Pero cómo hacemos la **decodificación**?

- x Necesitamos sí o sí saber el **código** usado

¿Pero cuál uso? **Latin1**

- x Está muy extendido
- x No se mantiene desde 2004
- x No representa todos los caracteres:

```
>>> print pi
```

```
π
```

```
>>> pi.encode("latin1")
```

```
...
```

```
UnicodeEncodeError: 'latin-1' codec can't encode  
character u'\u03c0'...
```

x Se utiliza un byte

- x Primeros 128 caracteres: **ASCII**
- x Segundos 128: para lenguajes de **Europa occidental**

¿Pero cuál uso? UTF-16

- x Soporta todos los caracteres
- x Pero tenemos que perder dos bytes por caracter (más el **BOM**!)
- x Incluso si utilizamos caracteres simples, como en esta linea.

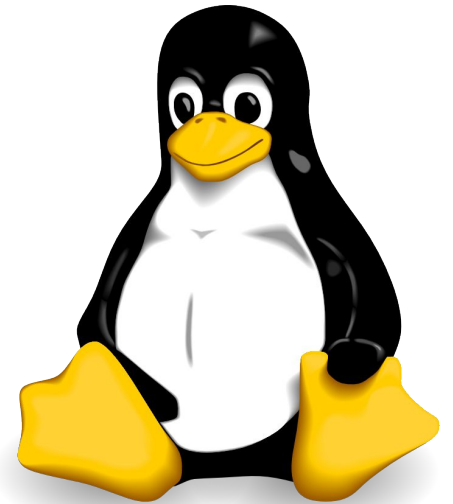
```
>>> u"esta".encode("utf16")  
'\xff\xfe\x00s\x00t\x00a\x00'  
>>> # FFFE 6500 7300 7400 6100
```

- x Utiliza **desde** 2 bytes
 - x Usa dos bytes para el **BMP**, pero puede usar 4 bytes
 - x Se justifica si **siempre** tengo caracteres *raros*
 - x **No es UCS-2** (el cual usa 2 bytes fijos)

¿Pero cuál uso? UTF-8

- x Es el estándar en **Linux**
- x **Soporta todo** el espacio Unicode
- x Utiliza uno, dos, tres y cuatro bytes, según necesite

```
>>> u"k".encode("utf8")
'k'
>>> u"ñ".encode("utf8")
'\xc3\xb1'
>>> u"썻".encode("utf8")
'\xec\x8f\x94'
```
- x Nunca se desactualizaría



uteefe-ocho

- x **8 bits** para los 128 caracteres ASCII

0-7F: 0zzzzzzz

11101010
10101000
01001011
10011011

- x **16 bits** para letras latinas y de otros idiomas

80-7FF: 110yyyyy 10zzzzzz

- x **24 bits** para el resto del BMP

800-D7FF y E000-FFFF: 1110xxxx 10yyyyyy 10zzzzzz

- x **32 bits** para el resto de Unicode

10000-10FFFF: 11110www 10xxxxxx 10yyyyyy
10zzzzzz

Encoding & Decoding

```
>>> pi  
u'\u03c0'
```

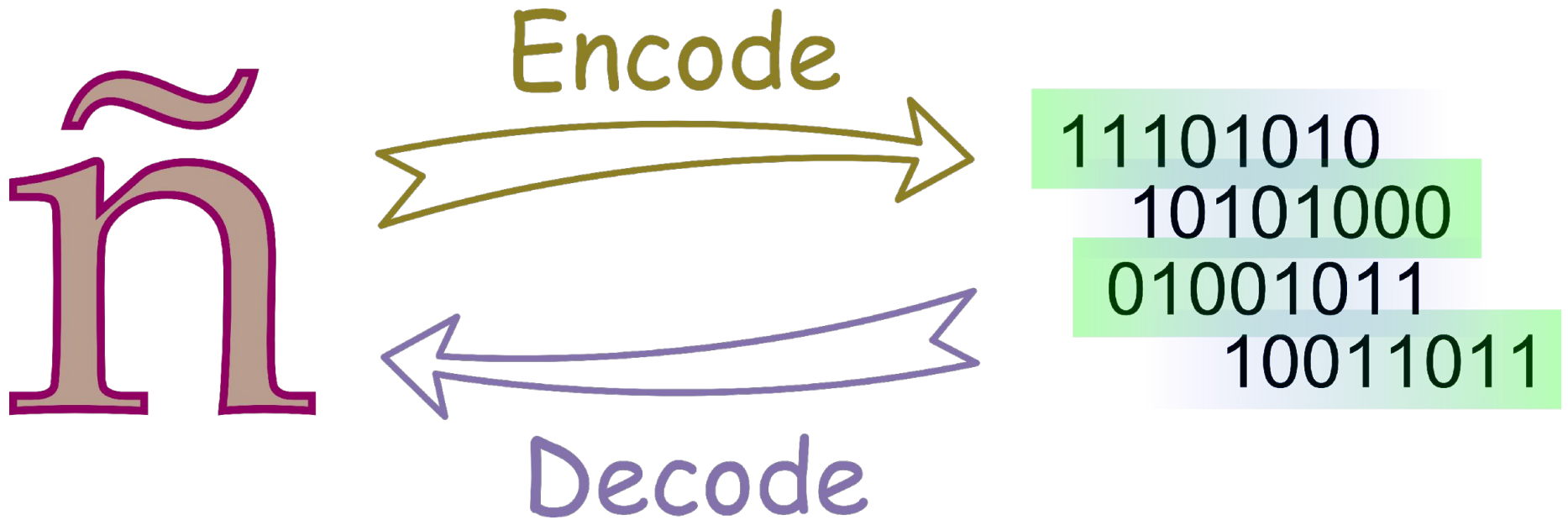
```
>>> d = pi.encode("utf8")  
>>> d  
'\xcf\x80'
```

```
>>> d.decode("utf8")  
u'\u03c0'
```

- x Python guarda las cadenas Unicode como bits
- x Usa una codificación interna en particular
- x Se decide al compilar Python: UCS-2 o UCS-4

Y dale...

¡Hasta que lo sepan de memoria!



Reglas de oro

- x **Internamente** siempre utilizar Unicode

```
>>> dato = 'm\xc3\xa1scara' # máscara en UTF8
>>> len(dato)
8
>>> print dato[:4]
más
>>> print dato.upper()
MÁSCARA
```

- x **Codificar y decodificar en los bordes**

- x encode/decode
- x codecs.open()

Siempre en los bordes

```
>>> nomapell = MiBDDWrapper(foo, bar)
>>> nomapell
'\xc5mal \xd6fwerman'
```

1. Leemos de
una
fuente externa

```
>>> completo = nomapell.decode("latin1")
>>> completo
u'\xc5mal \xd6fwerman'
>>> print completo
Åmal Öfwerman
```

2. Pasamos a
Unicode (decode)

```
>>> n, a = completo.split()
>>> tit = "%s, %s" % (a, n)
>>> print tit
Öfwerman, Åmal
```

3. Procesamos
internamente

```
>>> titok = tit.encode("utf8")
>>> titok
'\xc3\x96fwerman, \xc3\x85mal'
```

4. Pasamos a
bytes
(encode)

5. Dejamos el
resultado

```
>>> alHTML(titok)
```

Ejemplo!

Código fuente Unicode

- x Caracteres **Unicode en nuestro código**
- x El editor también debe traducir a bytes
 - x ¡Lo **codifica**!
- x El intérprete de Python debe saber leerlo
 - x ¡Lo **decodifica**!
- x Tiene que **saber qué encoding** se usó
 - x **#-*- coding: X -*-**

Cosas piolas

```
>>> unicodedata.name(pi)
'GREEK SMALL LETTER PI'
>>> print unicodedata.lookup("GREEK SMALL LETTER THETA")
θ
>>> print u"caño de desagüe".upper()
CAÑO DE DESAGÜE
>>> print pi, pi.upper()
π Π
>>> unicodedata.decomposition(u"ñ")
'006E 0303'
>>> print u"\u006e \u0303"
ñ
```



¡Muchas gracias!

¿Preguntas?

¿Sugerencias?

Facundo Batista

facundo@taniquetil.com.ar
<http://www.taniquetil.com.ar>



Licencia: Creative Commons
Atribución-NoComercial-CompartirDerivadasIgual 2.5 Argentina
http://creativecommons.org/licenses/by-nc-sa/2.5/deed.es_AR