Encoding, Random Numbers and GUIs

- Encoding of String Characters
- Randomness and Random Sampling
- Basic `tkinter` Widgets
- Event-Based `tkinter` Widgets

Character encodings

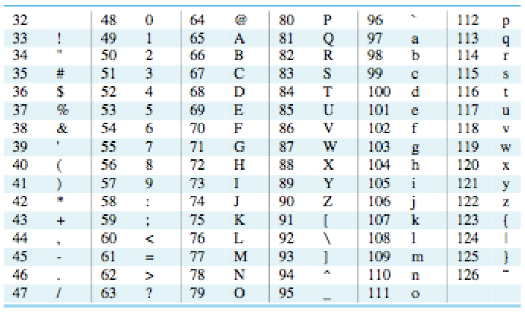
A string (`str`) object contains an ordered sequence of characters which can be any of the following:

- lowercase and uppercase letters in the English alphabet:
  `a b c … z` and `A B C … Z`
- decimal digits: `0 1 2 3 4 5 6 7 8 9`
- punctuation: `, . : ; ' " ! ?` etc.
- Mathematical operators and common symbols: `= < > + -`
  `/ * $ # % @ &` etc.
- More later

Each character is mapped to a specific bit encoding, and this encoding maps back to the character.

For many years, the standard encoding for characters in the English language was the American Standard Code for Information Interchange (ASCII)

ASCII



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

The code for a is 97, which is 01100001 in binary or 0x61 in hexadecimal notation

For many years, the standard encoding for characters in the English language was the American Standard Code for Information Interchange (ASCII)

The encoding for each ASCII character fits in 1 byte (8 bits)

Built-in functions `ord()` and `char()`

```
>>> ord('a')
97
>>> ord('?')
63
>>> ord('\n')
10
>>> chr(10)
'\n'
>>> chr(63)
'?'
>>> chr(97)
'a'
>>>
```

Function `ord()` takes a character (i.e., a string of length 1) as input and returns its ASCII code

Function `chr()` takes an ASCII encoding (i.e., a non-negative integer) and returns the corresponding character

Beyond ASCII

A string object contains an ordered sequence of characters which can be any of the following:

- lowercase and uppercase letters in the English alphabet: `a b c … z` and `A B C … Z`
- decimal digits: `0 1 2 3 4 5 6 7 8 9`
- punctuation: `, . : ; ' " ! ?` etc.
- Mathematical operators and common symbols: `= < > + - / * $ # % @ &` etc.
- Characters from languages other than English
- Technical symbols from math, science, engineering, etc.

There are only 128 characters in the ASCII encoding

Unicode has been developed to be the universal character encoding scheme

Unicode

In Unicode, every character is represented by an integer code point.

The code point is not necessarily the actual byte representation of the character; it is just the identifier for the particular character

The code point for letter `a` is the integer with hexadecimal value 0x0061

- Unicode conveniently uses a code point for ASCII characters that is equal to their ASCII code

With Unicode, we can write strings in

- english
- cyrillic
- chinese
- …

escape sequence `\u` indicates start of Unicode code point

```
>>> '\u0061'
'a'
>>> '\u0064\u0061d'
'dad'
>>>
'\u0409\u0443\u0431\u043e\u043c\u0438\u0440'
'Љубомир'
>>> '\u4e16\u754c\u60a8\u597d!'
'世界您好!'
>>>
```

String comparison, revisited

Unicode code points, being integers, give a natural ordering to all the characters representable in Unicode

Unicode was designed so that, for any pair of characters from the same alphabet, the one that is earlier in the alphabet will have a smaller Unicode code point.

```
>>> s1 = '\u0021'
>>> s1
'!'
>>> s2 = '\u0409'
>>> s2
'Љ'
>>> s1 < s2
True
>>>
```

Unicode Transformation Format (UTF)

A Unicode string is a sequence of code points that are numbers from 0 to 0x10FFFF.

Unlike ASCII codes, Unicode code points are not what is stored in memory; the rule for translating a Unicode character or code point into a sequence of bytes is called an encoding.

There are several Unicode encodings: UTF-8, UTF-16, and UTF-32. UTF stands for Unicode Transformation Format.

- UTF-8 has become the preferred encoding for e-mail and web pages
- The default encoding when you write Python 3 programs is UTF-8.
- In UTF-8, every ASCII character has an encoding that is exactly the 8-bit ASCII encoding.

Assigning an encoding to “raw bytes”

When a file is downloaded from the web, it does not have an encoding

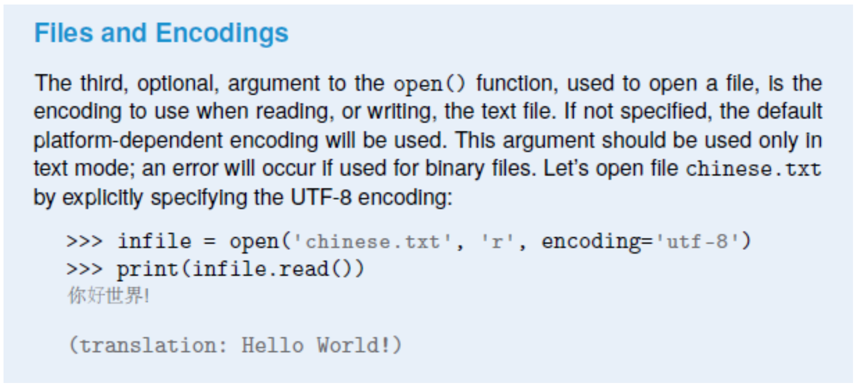
- the file could be a picture or an executable program, i.e. not a text file
- the downloaded file content is a sequence of bytes, i.e. of type `bytes`

The `bytes` method `decode()` takes an encoding description as input and returns a string that is obtained by applying the encoding to the sequence of bytes

- the default is UTF-8

```
>>> content
b'This is a text document\nposted on the\nWWW.\n'
>>> type(content)
<class 'bytes'>
>>> s = content.decode('utf-8')
>>> type(s)
<class 'str'>
>>> s
'This is a text document\nposted on the\nWWW.\n'
>>> s = content.decode()
>>> s
'This is a text document\nposted on the\nWWW.\n'
>>>
```

Opening Text Files with Encoding



Files and Encodings

The third, optional, argument to the `open()` function, used to open a file, is the encoding to use when reading, or writing, the text file. If not specified, the default platform-dependent encoding will be used. This argument should be used only in text mode; an error will occur if used for binary files. Let's open file `chinese.txt` by explicitly specifying the UTF-8 encoding:

```
>>> infile = open('chinese.txt', 'r', encoding='utf-8')
>>> print(infile.read())
你好世界!

(translation: Hello World!)
```

Randomness

Some apps need numbers generated "at random" (i.e., from some probability distribution):

- scientific computing
- financial simulations
- cryptography
- computer games

Truly random numbers are hard to generate

Most often, a pseudorandom number generator is used

- numbers only appear to be random
- they are really generated using a deterministic process

The Python standard library module `random` provides a pseudo random number generator as well useful sampling functions

Standard Library module `random`

Function `randrange()` returns a "random" integer number from a given range

Example usage: simulate the throws of a die

Function `uniform()` returns a "random" `float` number from a given range

range is from 1 up to (but not including) 7

```
>>> import random
>>> random.randrange(1, 7)
2
>>> random.randrange(1, 7)
1
>>> random.randrange(1, 7)
4
>>> random.randrange(1, 7)
2
>>> random.uniform(0, 1)
0.19831634437485302
>>> random.uniform(0, 1)
0.027077323233875905
>>> random.uniform(0, 1)
0.8208477833085261
>>>
```

Standard Library module `random`

Defined in module random are functions `shuffle()`, `choice()`, `sample()`, …

```
>>> names = ['Ann', 'Bob', 'Cal', 'Dee', 'Eve', 'Flo', 'Hal', 'Ike']
>>> import random
>>> random.shuffle(names)
>>> names
['Hal', 'Dee', 'Bob', 'Ike', 'Cal', 'Eve', 'Flo', 'Ann']
>>> random.choice(names)
'Bob'
>>> random.choice(names)
'Ann'
>>> random.choice(names)
'Cal'
>>> random.choice(names)
'Cal'
>>> random.sample(names, 3)
['Ike', 'Hal', 'Bob']
>>> random.sample(names, 3)
['Flo', 'Bob', 'Ike']
>>> random.sample(names, 3)
['Ike', 'Ann', 'Hal']
>>>
```

Exercise

Develop function `game()` that:

- takes integers `r` and `c` as input,
- generates a field of `r` rows and `c` columns with a bomb at a randomly chosen row and column,
- and then asks users to find the bomb

```
>>> game(2, 3)
Enter next position (format: x y): 0 2
No bomb at position 0 2
Enter next position (format: x y): 1 1
No bomb at position 1 1
Enter next position (format: x y): 0 1
You found the bomb!
```

Exercise

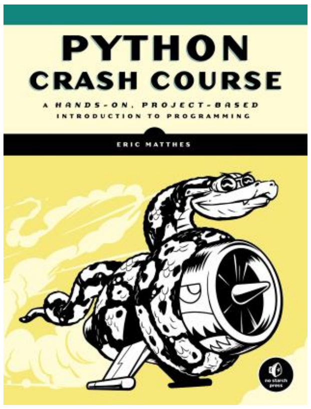
Develop function `game()` that:

- takes integers `r` and `c` as input,
- generates a field of `r` rows and `c` columns with a bomb at a randomly chosen row and column,
- and then asks users to find the bomb

```
import random
def game(rows, cols):
    'a simple bomb finding game'

    # generate a list of size rows*cols that contains
    # empty strings except for 1 'B' at some random index
    table = (rows*cols-1)*[''] + ['B']
    random.shuffle(table)

    while True:
        pos = input('Enter next position (format: x y): ')
        position = pos.split()
        # position (x, y) corresponds to index x*cols + y of table
        if table[int(position[0])*cols + int(position[1])] == 'B':
            print('You found the bomb!')
            break
        else:
            print('No bomb at position', pos)
```

Assignment 5

- The book contains 3 python projects. Implement one of these project on individual basis. The submission has to be made in hard copy format by 29th of December 2019. The choice of project is left to the student. The submission will be followed by a viva which will determine the students score.



GRAPHICAL USER INTERFACES

Graphical user interfaces (GUIs)

Almost all computer apps have a GUI

- A GUI gives a better overview of what an application does
- A GUI makes it easier to use the application.

A graphical user interface (GUI) consists of basic visual building blocks, called widgets, packed inside a standard window.

- widgets include buttons, labels, text entry forms, menus, check boxes, scroll bars, …

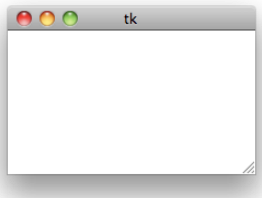
In order to develop GUIs, a developer will require a GUI application programming interface (API) that provides the necessary GUI toolkit. There are several GUI APIs for Python; in this text we use tkinter, a module that is part of Python's Standard Library.

Widget `Tk`

We introduce some of the commonly used `tkinter` widgets

Widget `Tk` represents the GUI window

```
>>> from tkinter import Tk
>>> root = Tk()
>>> root.mainloop()
```



As usual, the constructor creates the widget (i.e., GUI object) …
… but method `mainloop()` really starts the GUI

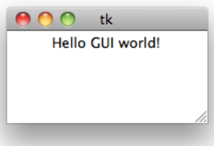
The window is currently empty; normally it contains other widgets

Widget `Label` (for displaying text)

The widget `Label` can be used to display text inside a window.

Option	Description
`master`	The master of the widget
`text`	Text to display on the widget

Method `pack()` specifies the placement of the widget within its master



```
>>> from tkinter import Tk, Label
>>> root = Tk()
>>> hello = Label(master = root, text = 'Hello GUI world!')
>>> hello.pack() # widget placed against top boundary of master (default)
>>> root.mainloop()
```

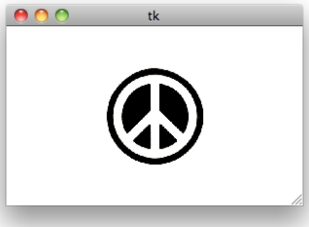
Widget `Label` (for displaying text)

The widget constructor has many options

Option	Description
`master`	The master of the widget
`text`	Text to display on the widget
`image`	Image to display
`width`	Width of widget (in pixels or characters)
`height`	Height of widget (in pixels or characters)
`relief`	Border style (FLAT, RAISED, RIDGE, …)
`borderwidth`	Width of border (no border is 0)
`background`	Background color (e.g., string "white")
`foreground`	Foreground color
`font`	Font descriptor (as a tuple
`padx, pady`	Padding added along the x- or y- axis

Widget `Label` (for displaying images)

The widget `Label` can be used to display images too

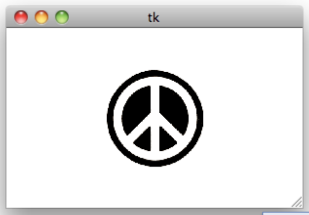


Option	Description
`master`	The master of the widget
`text`	Text to display
`image`	Image to display
`width`	Width of widget (in pixels or characters)
`height`	Height of widget (in pixels or characters)
`relief`	Border style (FLAT, RAISED, RIDGE, …)
`borderwidth`	Width of border (no border is 0)
`background`	Background color (e.g., string "white")
`foreground`	Foreground color
`font`	Font descriptor (as a tuple
`padx, pady`	Padding added along the x- or y- axis

`peace.py`

Widget `Label` (for displaying images)

The widget `Label` can be used to display images too

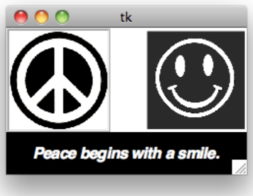


Option `image` must refer to an image in a format that `tkinter` can display. The `PhotoImage` class, defined in module `tkinter`, is used to transform a GIF image into an object with such a format.

```
from tkinter import Tk, Label, PhotoImage
root = Tk()
# transform GIF image to a format tkinter can display
photo = PhotoImage(file='peace.gif')

peace = Label(master=root,
              image=photo,
              width=300,    # width of label, in pixels
              height=180)   # height of label, in pixels
peace.pack()
root.mainloop()
```

Packing widgets

Method `pack()` specifies the placement of the widget within its master



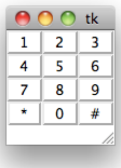
Option	Description
side	LEFT, RIGHT, TOP, BOTTOM,
fill	'both', 'x', 'y', or 'none'
expand	True or False

```
from tkinter import Tk, Label, PhotoImage, BOTTOM,
LEFT, RIGHT, RIDGE
root = Tk()

text = Label(root,
             font=('Helvetica', 16, 'bold italic'),
             foreground='white',
             background='black',
             pady=10, padx=25
             text='Peace begins with a smile.')
text.pack(side=BOTTOM)

peace = PhotoImage(file='peace.gif')
peaceLabel = Label(root,
                   borderwidth=3,
                   relief=RIDGE,
                   image=peace)
peaceLabel.pack(side=LEFT)

smiley = PhotoImage(file='smiley.gif')
smileyLabel = Label(root,
                    image=smiley)
smileyLabel.pack(side=RIGHT)

root.mainloop()
```

Arranging widgets into a grid

Method `grid()` is used to place widgets in a grid format

`pack()` and `grid()` use different algorithms to place widgets within a master; You must use one or the other for all widgets with the same master.



Options
`column`
`columnspan`
`row`
`rowspan`

```
from tkinter import Tk, Label, RAISED

root = Tk()
labels = [['1', '2', '3'],
          ['4', '5', '6'],
          ['7', '8', '9'],
          ['*', '0', '#']]

for r in range(4):
    for c in range(3):
        # create label for row r and column c
        label = Label(root,
                      relief=RAISED,
                      padx=10,
                      text=labels[r][c])
        # place label in row r and column c
        label.grid(row=r, column=c)

root.mainloop()
```

phone.py

Do you solve problems by just jumping in, willing to ignore the experience and wisdom of those that may have programmed solutions to problems very similar to yours? We learn from the past. Our ancestors discovered and invented ways of programming that we know call paradigms. We benefit from the knowledge they left us, even as we strive to create new paradigms ourselves.

PROGRAMMING PARADIGMS

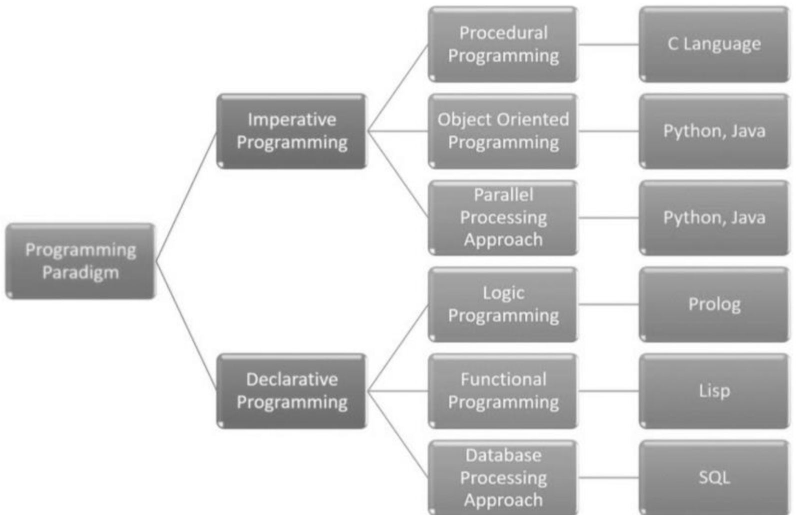
Programming Language Paradigms

- A programming paradigm is a paradigmatic style of programming (compare with a methodology which is a paradigmatic style of doing software engineering).
- A programming paradigm provides (and determines) the view that the programmer has of the execution of the program.
- The relationship between programming paradigms and programming languages can be complex since a programming language can support multiple paradigms.

Imperative and Declarative

- **Imperative programming** is a programming paradigm that uses statements that change a program's state.
- **Declarative programming** is a programming paradigm … that expresses the logic of a computation without describing its control flow.

Programming Language Paradigms



Imperative

– Procedural
  - Characterized by sequential instructions
  - A program in which statements are grouped into a hierarchy of subprograms
  - Fortran, C, C++

– Object-oriented model
  - Program consists of a set of objects and the interactions among the objects
  - Python, Java, Smalltalk, Simula

Declarative

– Functional
  - Based on the mathematical concept of a function
  - Lisp, Scheme, and ML

– Logic
  - Based on principles of symbolic logic
  - Types of statements
    - declares facts about objects and relationships
    - defines rules about objects
    - asks questions about objects
  - PROLOG

PROLOG

- Pets to owners

```
– owns(mary,bo).
– owns(ann,kitty).
– owns(bob,riley).
– owns(susy,charlie).

– ?-owns(mary,bo)
– yes
– ?-owns(bo,mary)
– no
– ?-owns(susy,bo)
– no
```

States facts

Asks questions

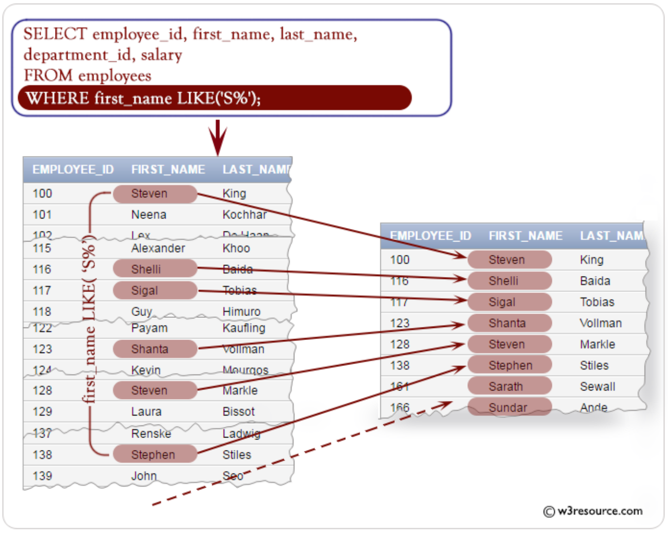
PROLOG

- `?-owns(ann, Cat).`
- `Cat = kitty`

- `?-owns(Name,charlie).`
- `Name = susy`

Upper case is variable; lower case is constant

Database (SQL)

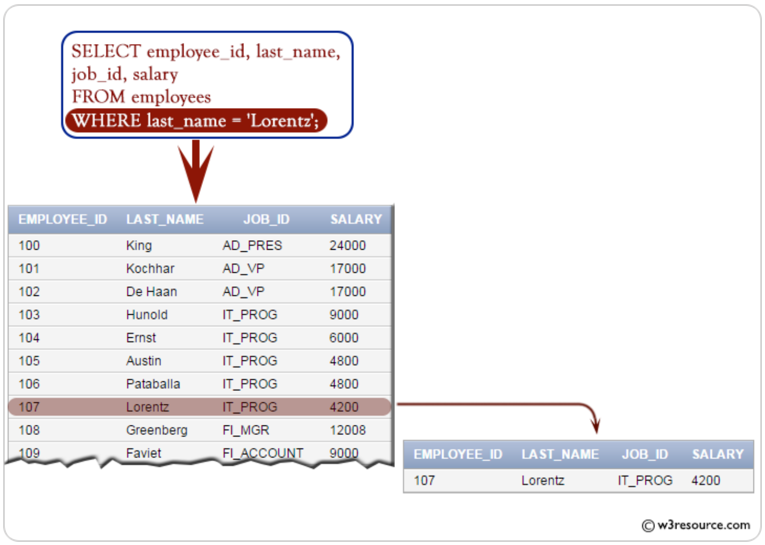


```
SELECT employee_id, first_name, last_name,
department_id, salary
FROM employees
WHERE first_name LIKE('S%');
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAM
100	Steven	King
101	Neena	Kochhar
102	Lex	De Haan
115	Alexander	Khoo
116	Shelli	Baida
117	Sigal	Tobias
118	Guy	Himuro
122	Payam	Kaufling
123	Shanta	Vollman
124	Kevin	Mourgos
128	Steven	Markle
129	Laura	Bissot
137	Renske	Ladwig
138	Stephen	Stiles
139	John	Seo

EMPLOYEE_ID	FIRST_NAME	LAST_NAM
100	Steven	King
116	Shelli	Baida
117	Sigal	Tobias
123	Shanta	Vollman
128	Steven	Markle
138	Stephen	Stiles
161	Sarath	Sewall
166	Sundar	Ande

Database (SQL)



```
SELECT employee_id, last_name,
job_id, salary
FROM employees
WHERE last_name = 'Lorentz';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
103	Hunold	IT_PROG	9000
104	Ernst	IT_PROG	6000
105	Austin	IT_PROG	4800
106	Pataballa	IT_PROG	4800
107	Lorentz	IT_PROG	4200
108	Greenberg	FI_MGR	12008
109	Faviet	FI_ACCOUNT	9000

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
107	Lorentz	IT_PROG	4200

Event-driven programming

When a GUI is started with the `mainloop()` method call, Python starts an infinite loop called an event loop

```
while True:
    1. wait for an event to occur
    2. run the associated event handler
```

Event-driven programming is the programming approach used to build applications whose execution flow is determined by events and described using an event loop