PYTHON
Object-Oriented Programming focuses on creating reusable patterns of code, not procedural with sequenced instructions
Objects are all python data, everything in python is an object and every object has an identity, a type and a value

ATTRIBUTES and IDENTIFIERS
Attributes - obj variables and function/methods dir(obj). Identifiers - obj names, class start with uppercase, all other lower

STATEMENTS and SUITES
Multiple statements (instructions) on single line - use ; → a=10; b=20; c=a+b; print c
Single statemet on multiple lines – use \ (not required within [], {} and ()) → c = a+\
b
Comments start with # → # this is a remark
Complex statemets like if, while, def, … start with a header statement with a colon (:) followed by a statement or suite (code block) that needs to be indented by the same amount of tabs/spaces → if age =18:
print "you are an adult"

VARIABLES and INTERPRETERS
int (integers) / float (real-decimal) / long (long, octal, hexa) / complex (complex) / bool (boolean-true+false) / str (strings)
str to code: eval(str,{glo},{loc}) {}-restrict code | exec(str,{glo},{loc}) → eval('abs(a)', {'__builtins__': None}, {'a': a, 'abs': abs})

OPERATORS
+,-,*,/,**,% - add,subtract,multiply,divide,exponential,remainder | >,<,<=,>=,!=(<>),== - comparison | =,+=,-=,*=,/= - assign
object.attribute – access att from obj | ("%s %d %.nf" %(str,int,float)) – str | x[in] – Index | and, not, or | in, not in | is, not is
→ c+=a equal to c=c+a | print ("%d %s %.2f" %(5,"dobro",2.501)) | text="SUPER"; text.lower(); text[2:3]; "P" in text (true)

STRINGS, LISTS, TUPLES, SETS and DICTIONARIES
Strings str "" → tx="hi you"; tx[0]; tx[1:4]; len(tx); tx.replace("hi","hey"); tx.split(" "); tx.upper() | \n-new line; \"-" | str(obj)
Lists list [] sequence of data → ls=["cat","dog",5]; del ls[1]; an[0]; ls.sort(); ls.append("fish"); ls.insert(2,"owl"); ls.remove(5)
Tuples tuple () immutable list → tu=(23,7,"car","bike",7); tu[2]; tu.reverse(); tu.count(7); tu.index("bike") | tuple(obj)
Set set {} unique unindexed list → st={"ane","zack"}; len(st); st.add("john"); st.discard ("ane"); st.clear() | st1.difference(st2)
Dictionaries dict {} key-value pairs → dc={"red":3,"blue":8}; dc.keys(); dc.values(); dc.items(); dc["red"]; dc["green"]=13

IF, ELIF and ELSE
Decison making process based on conditions with colons (:) followed by statement/suite

```
→   age=input ("age?")
    if age<18: print "you cant vote"
    elif 18<=age<=70: print "you can vote"
    else: print "you can old man"
```

WHILE and FOR
Loops based on condition (while) or sequence (for) with colons (:). break terminates, continue moves to next iteration

```
→   a=0
    while a<10:
        a+=1
        if a==5: continue
        print a
    else: print "done without 5"
```

```
→   pot2=[1,2,4,8,16,32]
    for nr in pot2:
        if nr==7:
            print "7 found"
        break
    else: print "7 not in list"
```

FUNCTIONS
def with name, parm/args (..,..,..) and colon (:) following by statement/suit to perform tasks and ends with return
(*args) varied amount of non-keyworded arguments / (**kwargs) variable length keyworded dictionary
Private (__var cannot be accessed) and local variables are set within functions, global outside or defined → global var
Bult-in: abs(x),dict(),dir(obj),float(x),int(x),len(s),list(),max(s),min(s),open(f),range(x),round(x,n),sum(s),tupple(),type(obj),..

```
→   def pytha (sid1,sid2):
        sqahip=(sid1**2+sid2**2)
        return sqahip
    a=input("side 1 ?"); b=input("side 2 ?"); c=pytha(a,b); print "Square of the hypotenuse = ",c
```

CLASSES
Object template or construct ("blueprint"), attributes the object support accessed via dot(.), user defined type(obj).
__init__ object constructor initial method, self represent the objects(instances) names and instances are class realizations

```
→   class Bankacc():
        bonus=10
        def __init__ (self, address, balance):
            self.address=address
            self.balance=balance
        def deposit (self, amount):
            self.balance+=(amount+Bankacc.bonus)
            return self.balance
    john=Bankacc("Lx",80); ines=Bankacc("Pt",50); n=input ("Name?"); v=input("Value?"); print n.deposit(v)
```

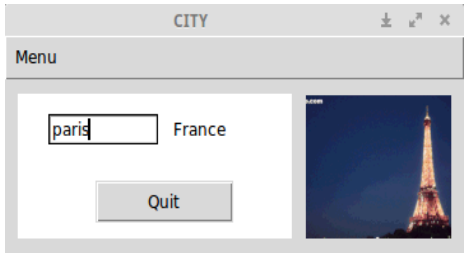
Bankacc is the class, variable bonus and method deposit are the attributes, john and ines are class instances/objects

TXT and CSV files
Open file → f=open("./hello.txt","a+") | Read → f.readlines() | Write → f.writelines("abc 10") | Close → f.close()
from csv import * | c=open("./file.csv","r/a/w") | re=reader(c,delimiter=",") | ls=list(re) | wr=writer(c); wr.writerow(s)=(4,5,9)

EXCEPTIONS
Error handling - try: a statement/suite, except: perform tasks on error, else: tasks if OK and finally: tasks either way
→ try: f=open("test.txt"); f.write("car") | except: print ("file problem") | else: print ("car added") | finally: f.close()

MODULES
Use extra predefined code → import math as mt; print mt.cos(1.5) | from math import *; print cos(1.5)
Built-in: csv,datetime,email,html,http,ipaddress,math,mimetypes,os,random,re,socket,sys,statistics,tkinter,zipfile,..

THINKER GUI
Window splitted in frames with widgets (Label,Entry,Button,..). Types: grid(table like implementation),pack,place
Better min 3 code blocks: mainWin+Menus – Frames – Widgets | → from Thinker import */ at end → w.mainloop()
Window → w=Tk(); w.title("P"); w.geometry("widthxheight+posx+posy"); w.minsize(widthxheight); w.grid_row/columnco...
Menus → m=Menu(w); m1=Menu(m); w.config(menu=m); m.add_cascade(label="M",menu=m1); m1.add_command(..
Frames → f=Frame(w,o='..',o=.., ..) o-width/height/(i)padx y-margins (i-internal)/bd (border)/relief (3d-flat,raised,sunken)/
cursor (circle,dotbox,exchange,fleur,plus,target,tcross,watch,..)/bg,highlightcolor,highlightbackground (colors)
Variables → v1=StringVar()/BooleanVar()/DoubleVar()/IntVar() | v1.trace("y",command) y-w/r/u | v1.set("hi") | v2=v1.get()
Widgets → d=x(f,o='..',o=.., ..) x-Message/Label/Listbox/Entry/Spinbox/Button/Menubutton/Radiobutton/Checkbutton/
Scrollbar/Scale o-state(normal,disabled,active)/width/height/(i)padx y/bd/relief/cursor/bg,fg,activebackground
foreground,highlightcolor background/text/font/justify/anchor(W,E,N,S)/image/textvariable/variable/command
Widget Events → x.bind("<y>",command) y-char/Shift-a,Alt-a,Control-a (a-char,_L-any)/Key/KeyRelease/Return/FocusIn Out
(keyboard focus)/Bb-Motion, (Double-)Button-b, ButtonRelease-b (mouse b-1,2,3)/Enter or Leave (mouse-the widget)
Grids → x.grid(row/rowspan=n,column/columnspan=n,o=.., ..) o-(i)padx y/sticky (W,E,N,S)/weight (0,1,.. resize factor) |
x.grid_row/columnconfigure(n,o=.., ..) n-inside row/column nr o-(i)padx y/minsize (avoid shrink to widget size)/weight
→




```
#!/usr/bin/env python2
import os; from Tkinter import *
# classes + instances
class Capit():
	def __init__(self, country, image):
		self.country=country
		self.image=image
rome=Capit('Italy','rom.gif');paris=Capit('France','par.gif');oslo=Capit('Norway','osl.gif')
# functions (d_path - pyinstaller add-data path)
def d_path(relative_path):
	if hasattr(sys, '_MEIPASS'):
		return os.path.join(sys._MEIPASS, relative_path)
	return os.path.join(os.path.abspath("."), relative_path)
def update(*args):
	getvar = entvar.get()
	if getvar not in ["rome","paris","oslo"]:
		entvar.set(""); lblvar.set(""); return
	cityvar = eval(getvar, {'__builtins__': None},{"rome": rome, "paris": paris, "oslo": oslo})
	lblvar.set(cityvar.country);cityph=PhotoImage(file=(d_path(cityvar.image)));lb2.image=cityph;lb2.configure(image=cityph)
	return
# main window centered + menus
win=Tk(); win.title('CITY'); x_pos=(win.winfo_screenwidth()/2-180); y_pos=(win.winfo_screenheight()/2-60); win.geometry("360x120+%d+%d"%
(x_pos,y_pos)); win.minsize(360,120)
win.grid_columnconfigure(0, weight=1); win.grid_columnconfigure(1, weight=1); win.grid_rowconfigure(0,weight=1)
mnu=Menu(win); win.config(menu=mnu); mn1=Menu(mnu); mnu.add_cascade(label='Menu',menu=mn1); mn1.add_separator();
mn1.add_command(label="Exit",command=win.quit)
# frames (only minsize required)
lft=Frame(win,bg="white"); lft.grid(row=0,column=0,padx=10,pady=10)
lft.grid_rowconfigure(0,minsize=50);lft.grid_rowconfigure(1,minsize=50)
lft.grid_columnconfigure(0,minsize=115);lft.grid_columnconfigure(1,minsize=115)
rgt=Frame(win,bg="black"); rgt.grid(row=0,column=1,padx=(0,10),pady=10)
rgt.grid_rowconfigure(0,minsize=100)
rgt.grid_columnconfigure(0,minsize=100)
# widgets
entvar=StringVar(); ent=Entry(lft, textvariable=entvar, width=10); ent.bind("<Return>", update)
lblvar=StringVar(); lb1=Label(lft,textvariable=lblvar, bg="white")
bot=Button(lft,text="Quit",width=10,command=win.quit)
cityph=PhotoImage(file=(d_path(rome.image))); lb2=Label(rgt, image=cityph)
ent.grid(row=0,column=0,sticky=E, padx=5)
lb1.grid(row=0,column=1,sticky=W, padx=5)
bot.grid(row=1,columnspan=2)
lb2.grid(row=0,column=0,sticky=NSEW)
win.mainloop()
```

Git, App and Deb package
>git o | o-init/add/reset <files>(*;.)/commit -m 'vs'/status,log (--stat -h;-p)/diff/checkout/push,pull,clone | del→rm -rf .git
>pyinstaller --o .. prog.py | o-onefile/add-data 'file.txt:.'(csv,jpg,..) | outs: dist(app)+build+.spec file | run→./app (755)
>deb | mkdir -p app/ 1-DEBIAN 2-usr/ 2.1-bin 2.2-share/ 2.2.1-applications 2.2.2-icons/hicolor/scalable/apps |
1-nano control .. | 2.1-copy app | 2.2.1-nano app.desktop .. | 2.2.2-app.svg | dpkg-deb --build app