OBJECT ORIENTATION - PLAYING WITH OBJECTS

WHY DO WE NEED OBJECTS?

They allow to create a simple representation of any complex scenario in in the computer.

REMINDER: FORMAL DEFINITION OF AN OBJECT

Objects in "object-oriented programming" are essentially data structures together with their associated processing routines.
A object thus has:

- data structures
- processing routines (=called methods; similar to functions)

Reminder: Modelling a cow

Which data-structures (which information does a cow have)?

- age
- milk yield
- ?? any other ideas

Which formal routines (what can you do with a cow)?

- feed()
- milk()
- ??

CLASS VS INSTANCE

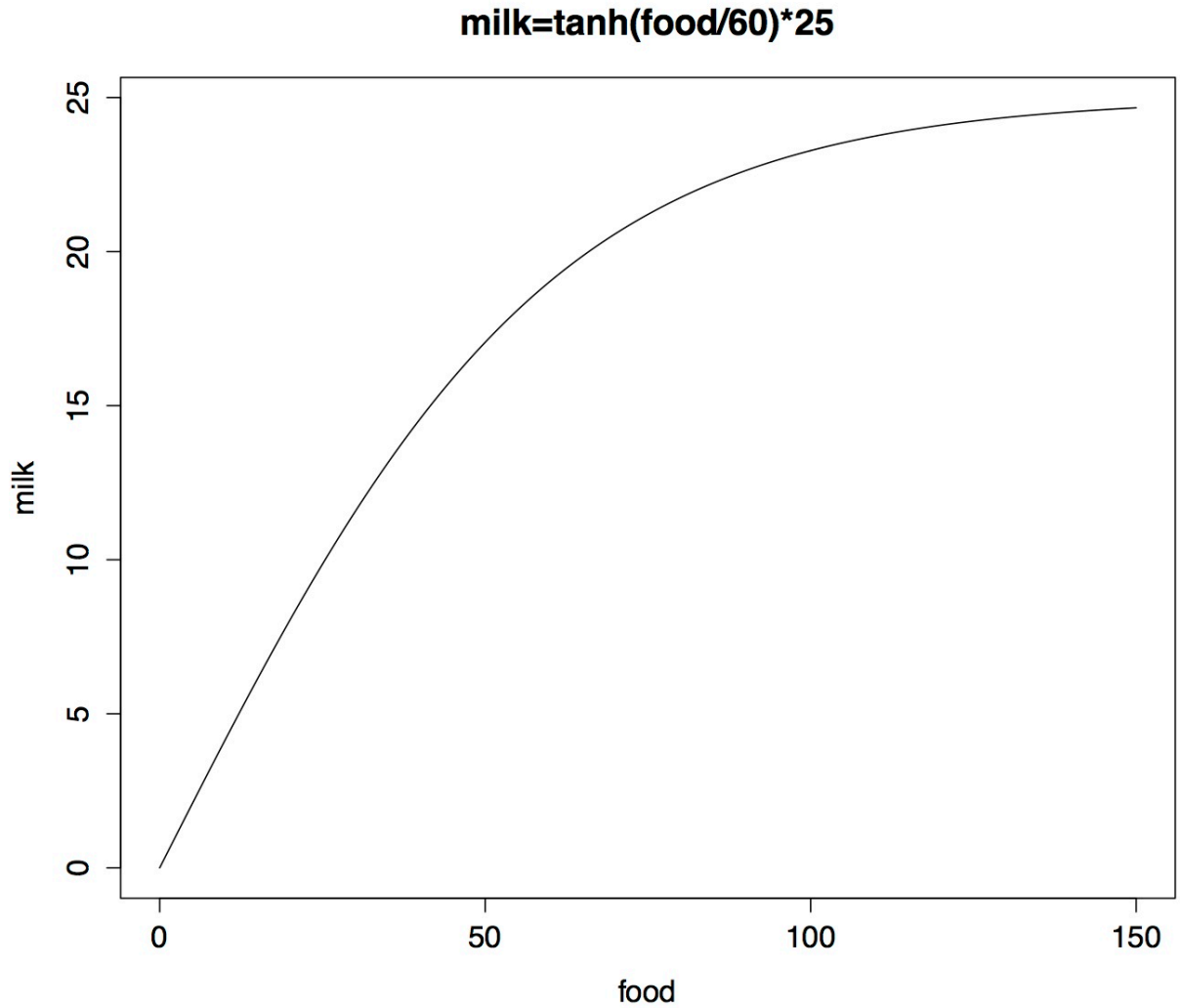
Class
All cows adhere to the same template; the all have a age, a milk yield and so on, this template is called ”class”.
Instance
One specific cow adhering to the template (class). Thus the cow named ”Resi” is an instance, a cow named ”Pepi” is an instance and so on.
To summarize, a class ’cow’ represents the general concept of a cow, an instance of a ’cow’ is one particular cow.

FARMVILLE3

Cow info

A Swiss cow consumes about 70-100 kg grass per day and produces about 20-25kg milk

RELATIONSHIP BETWEEN FOOD AND MILK?

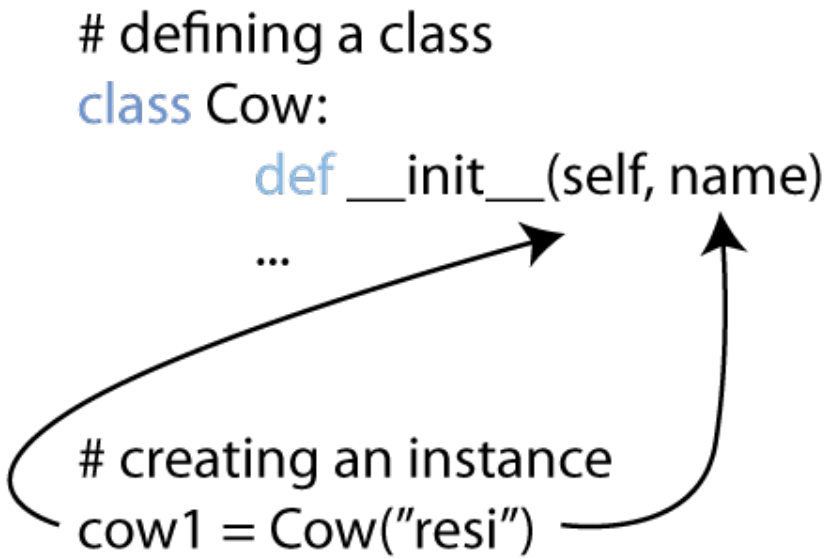


A cow!

```
1  import math
2
3  class Cow:
4      def __init__(self,name):
5          self.name=name
6          self.foodinbelly=0.0
7
8      def feed(self,amount):
9          # we just add the food to belly-content
10         # multiple feeding would be possible
11         self.foodinbelly+=float(amount)
12
13     def askName(self):
14         return self.name
15
16     def getMilk(self):
17         food=self.foodinbelly
18         # compute the milk-yield
19         milk=math.tanh(food/60.0)*25
20         # all the food has been transformed to milk
21         # reset to zero
22         self.foodinbelly=0
23         return milk
```

COW CONSTRUCTOR

A constructor is a special method (__init__) that is called only once when an instance of a class is created (eg the cow resi). In Python, the newly created instance will always be passed as first parameter to the constructor.

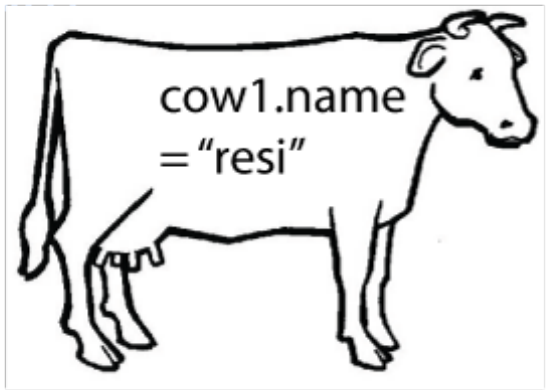


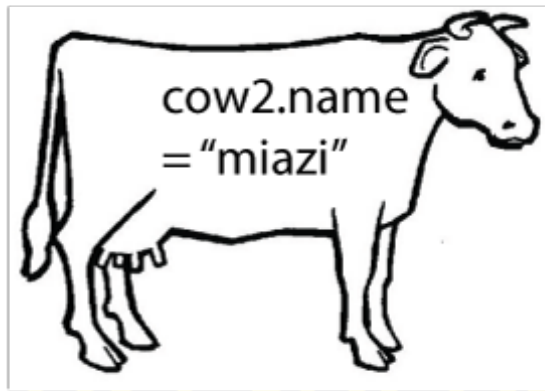
Cow instances

```
# defining a class
class Cow:
        def __init__(self, name)
        self.name = name
        self.foodinbelly = 0.0

# creating an instance
cow1 = Cow("resi")
cow2 = Cow("miazi")
print(cow1.name)
print(cow2.name)
```

Instances of cows:





USING A COW

```
1  import math
2
3  # class Cow...
4
5  mycow = Cow("resi")
6  print(mycow)
7  print("Cow name ", mycow.askName())
8
9  mycow.feed(20)
10 mymilk = mycow.getMilk()
11 print("Ha, i got {0} liters milk".format(mymilk))
12 # format is a method of string (like split and ?)
13 # it replaces {0} with the first argument
14 # {1} with the second and so on ...
```

MINITASK; IS THE COW HUNGRY?

```
1 # add a new method to the cow
2 # with the name isHungry
3 # is the belly empty?
4 # true or false
5
6 # ask your cow before and after feeding
7 # before and after milking
```

Unser Lagerhaus

LAGERHAUS

```
1  import math
2  import random
3  ### COW ###
4
5  class Lagerhaus:
6      def __init__(self,cowprize,foodprize,milkprize):
7          self.cowprize = cowprize
8          self.foodprize = foodprize
9          self.milkprize = milkprize
10         self.cownames=["miazi", "resi", "heidi", "parishilton"]
11
12     def buyCows(self, money):
13         cowcount=int(money/self.cowprize)
14         cows=[]
15         for i in range(0,cowcount):
16             cow=Cow(random.choice(self.cownames))
17             # random.choice pics a random element form a list
18             cows.append(cow)
19         return cows
20
21     def buyFood(self, money):
22         food=int(money/self.foodprize)
23         return food
24
25     def sellMilk(self, milk):
26         money=milk*self.milkprize
27         return money
28
29     def status(self):
30         print("The cow costs {0}; the food {1}; the milk is worth {2};".format(self.
       cowprize,self.foodprize,self.milkprize))
```

MINITASK; USE AND EXTEND THE LAGERHAUS

```
1 # use the Lagerhaus
2 # a.) creat an instance of a Lagerhaus
      lh_obertupfing = ...
3 # b.) buy a cow
4 # c.) buy ten cows and print their names
5 # d.) sell some milk and print the obtained
      money
6
7 # What else could a Lagerhaus do?
8 # e.) add at least two methods to the Lagerhaus
```

Farm

The Farm - Part 1

```
 1  import math
 2  import random
 3  ### COW ###
 4  ### LAGERHAUS###
 5  class Farm:
 6      def __init__(self, lagerhaus, startingbudget):
 7          self.lagerhaus = lagerhaus
 8          self.stable = [] # the stable is empty
 9          self.foodstored = 0
10          self.money = startingbudget
11
12      def buyCows(self, money):
13          if(money > self.money):
14              money = self.money
15              # if we don't have enough, just spend the rest
16          newcows=self.lagerhaus.buyCows(money)
17          self.money -= money # so we spent some money
18          self.stable.extend(newcows)
19          print("Bought {0} new cows".format(len(newcows)))
20
21      def buyFood(self, money):
22          if(money > self.money):
23              money=self.money
24          newfood=self.lagerhaus.buyFood(money)
25          self.money -= money
26          self.foodstored += newfood
27          print("Bought {0} new food".format(newfood))
```

THE FARM - PART 2

```
....

def feedCows(self,amount):
    if amount > self.foodstored:
        amount = self.foodstored
        # if there is not enough food left, just use the rest
    amountpercow=float(amount)/len(self.stable) # number of cows
    self.foodstored-=amount
    for cow in self.stable:
        cow.feed(amountpercow)

def sellMilk(self):
    milk=0
    for cow in self.stable:
        milk+= cow.getMilk()
    newmoney=self.lagerhaus.sellMilk(milk)
    self.money+=newmoney
    print("Sold {0} milk, got {1} money".format(milk,newmoney))


def status(self):
    print("Your farm has {0} cows; and {1} kg food; and {2} money".format(len(self
  .stable),self.foodstored,self.money))
```

OBJECT ORIENTED DESIGN - TRUE ART?

So the farm has a Lagerhaus...was this a good design decision?

Relationship between farm and Lagerhaus

So in a real live situation

- there may not be a Lagerhaus at all
- the farmer may decide to switch to a different Lagerhaus, with different prices
- what else?
- would changing prices cause an error in the game? what about robustness?

Rule of thump: try to model the relationships like in the real world, a farm can exist even without Lagerhaus.

The game: Farmville3

FARMVILLE3

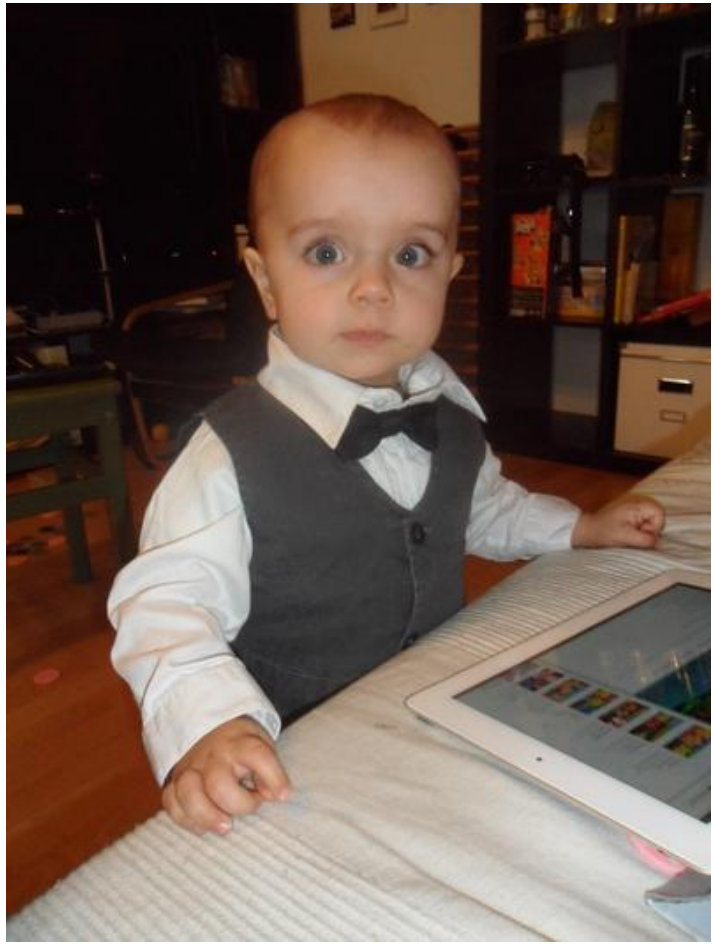
```
1  import math
2  import random
3  ### COW ###
4  ### LAGERHAUS ###
5  ### FARM ###
6
7  meinLagerhaus=Lagerhaus(250,1,14)
8  myFarm=Farm(meinLagerhaus,3000)
9
10 for i in range(0,10):
11     print ("Round {0}".format(i+1))
12     meinLagerhaus.status()
13     myFarm.status();
14     money=int(input("For how much money do you want to buy cows? "))
15     myFarm.buyCows(money)
16     myFarm.status()
17     money=int(input("For how much money do you want to buy food? "))
18     myFarm.buyFood(money)
19     print("Feeded all the food to cows; Selling the resulting milk")
20     myFarm.feedCows(myFarm.foodstored)
21     myFarm.sellMilk()
22     myFarm.status()
23     print("\n\n")
24
25 print("Game over; you have {0} money".format(myFarm.money))
```

LAST TASK: BE CREATIVE :)

```
1 # now it's your tern to apply what
2 # you learned during this lecture
3
4 # a.) think of something you'd like to model
5 # b.) write a similar tool
6 # c.) this will be an important part of the
      final exam
7 # d.) every script should be unique :)
```

GUI-PROGRAMMING: BRINGING COLOR INTO PROGRAMMING

What are the advantages of a GUI?



- Every child can use it....that's actually my child using an iPad with 16 months..
- Interactive data exploring is possible, eg. genome browsers

What are the disadvantages of a GUI?

- automatization is more difficult
- incorporation into existing pipelines is impossible
- programming is more timeconsuming, especially the boring part of programming

HELLO WORLD WITH TKINTER

```
1  from tkinter import *
2  # create the root widget
3  # ordinary window with title bar
4  # only one root widget per program
5  # always create first
6  root = Tk()
7  # create label as child of root
8  w = Label(root,text="Hello, World!")
9  # label must fit the size and make itself
       visible
10 w.pack()
11 # start the tool
12 root.mainloop()
13 # press F5
```

Nice, but that's not going to impress the ladies

You have a nice tool with GUI, but to start it with Idle or from the command line is not impressive.

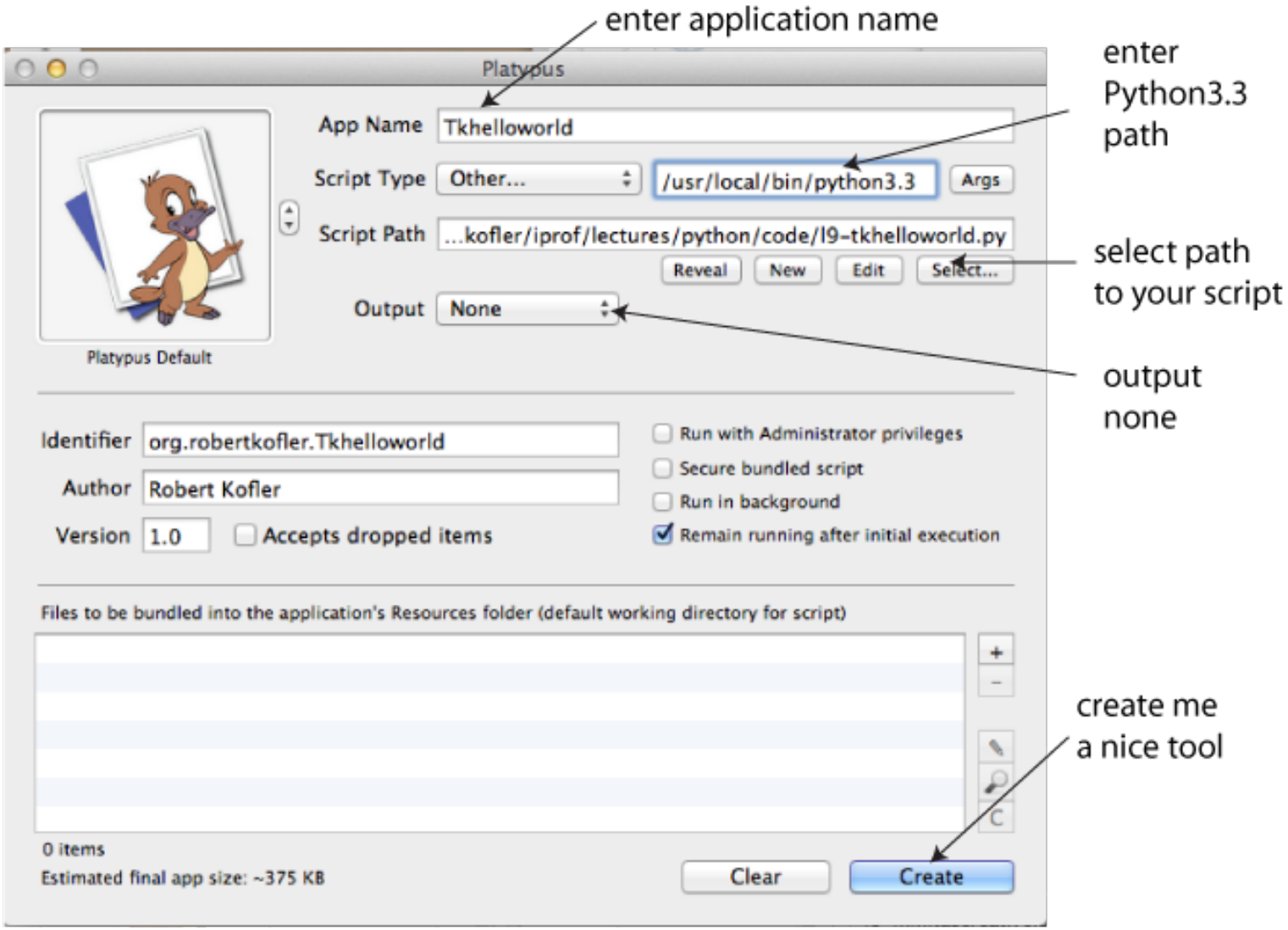
$\Rightarrow$ you should at least be able to start the app in Finder by double-clicking

How to convert the script to a Mac app?

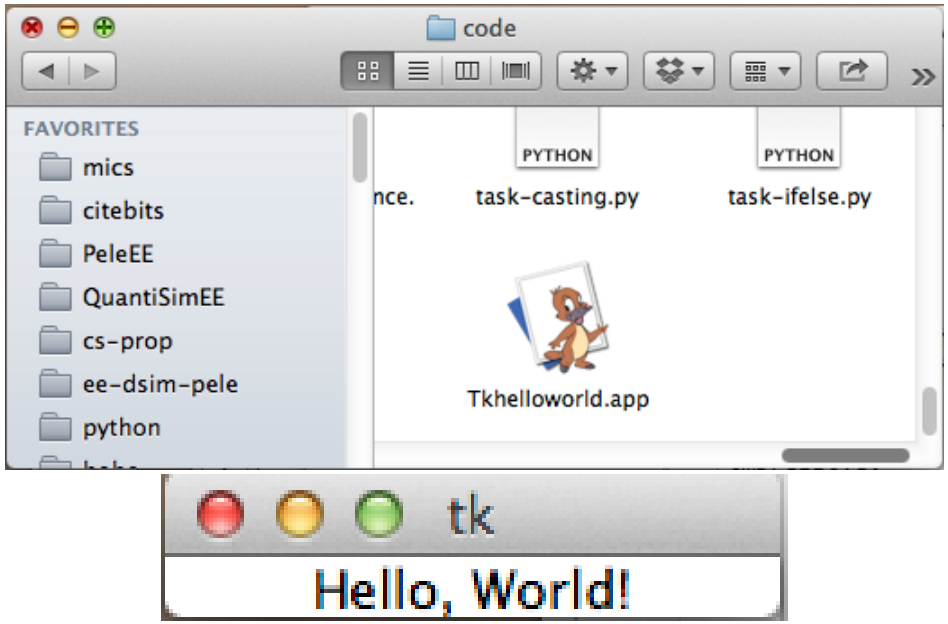
Convert the Python script with Platypus into a Mac application:
http://sveinbjorn.org/platypus

$\Rightarrow$ download, install and start Platypus

PLATYPUS


enter application name
enter Python3.3 path
select path to your script
output none
create me a nice tool
Platypus
App Name
Tkhelloworld
Script Type
Other...
/usr/local/bin/python3.3
Args
Script Path
...kofler/iprof/lectures/python/code/l9-tkhelloworld.py
Reveal
New
Edit
Select...
Output
None
Platypus Default
Identifier
org.robertkofler.Tkhelloworld
Author
Robert Kofler
Version
1.0
Accepts dropped items
Run with Administrator privileges
Secure bundled script
Run in background
Remain running after initial execution
Files to be bundled into the application's Resources folder (default working directory for script)
0 items
Estimated final app size: ~375 KB
Clear
Create

VOILA, YOUR OWN MAC APP



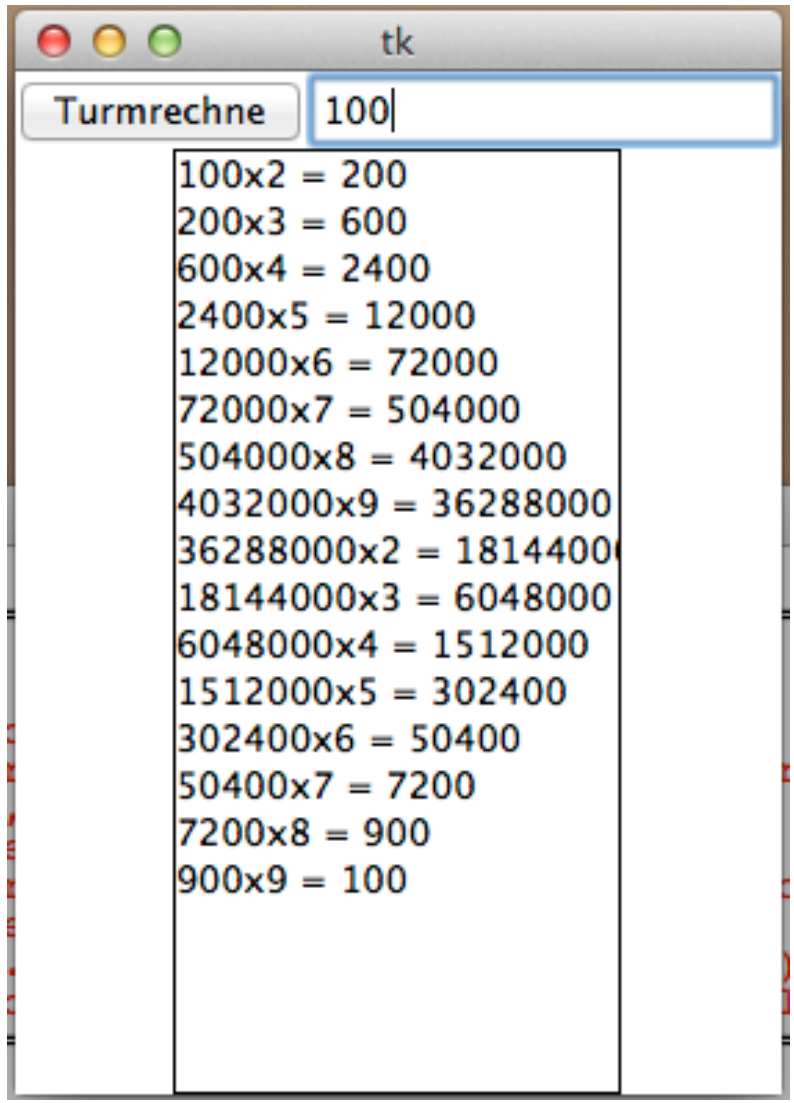
More complex app with a button

```
1  from tkinter import *
2  class App:
3      def __init__(self,master):
4          # frame is a container, holding other widgets
5          frame = Frame(master)
6          frame.pack() # make the frame visible
7          # create a button with some text
8          # when pressed execute function update, remember sort?
9          self.button=Button(frame,text="say hello", command=self.
       update)
10         self.button.pack(side=LEFT)
11         #  create a Label right of the button
12         self.lab = Label(frame,text="you suck")
13         self.lab.pack(side=LEFT)
14
15     def update(self):
16         self.lab.config(text="sorry, you are great")
17
18 root=Tk()
19 app=App(root)
20 root.mainloop()
```

TURMRECHNER

```
1  from tkinter import *
2  class App:
3      def __init__(self,master):
4          frame=Frame(master)
5          frame.pack() # make the frame visible
6          self.button=Button(frame,text="Compute",command=self.compute)
7          self.button.pack(side=LEFT)
8          #entry allows to enter text
9          self.ent=Entry(frame)
10         self.ent.pack(side=LEFT)
11         frame2=Frame(master)
12         frame2.pack()
13         self.listbox=Listbox(frame2,height=20)
14         self.listbox.pack()
15
16     def compute(self):
17         self.listbox.delete(0,END) # reset content of listbox
18         val=int(self.ent.get())
19         for i in range(2,10):
20             self.listbox.insert(END,"{0}x{1} = {2}".format(val,i,val*i))
21             val=val*i
22         for i in range(2,10):
23             self.listbox.insert(END,"{0}x{1} = {2}".format(val,i,int(val/i)))
24             val=int(val/i)
25
26
27 root=Tk()
28 app=App(root)
29 root.mainloop()
```

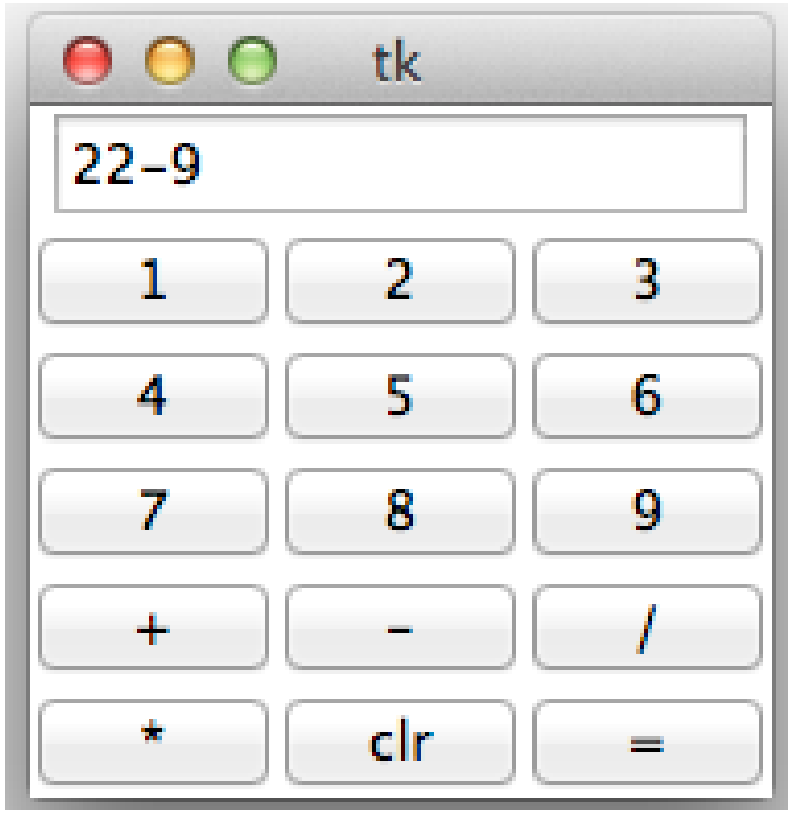
TURMRECHNER IN ACTION



MY FIRST POCKET CALCULATOR

```
 1  from tkinter import *
 2  from tkinter import Button as B
 3  class App:
 4      def __init__(self,m):
 5          e=Entry(m)
 6          e.grid(columnspan=3)
 7          B(m,text=1,command=lambda:e.insert(END,1),width=4).grid(row=1,column=0)
 8          B(m,text=2,command=lambda:e.insert(END,2),width=4).grid(row=1,column=1)
 9          B(m,text=3,command=lambda:e.insert(END,3),width=4).grid(row=1,column=2)
10          B(m,text=4,command=lambda:e.insert(END,4),width=4).grid(row=2,column=0)
11          B(m,text=5,command=lambda:e.insert(END,5),width=4).grid(row=2,column=1)
12          B(m,text=6,command=lambda:e.insert(END,6),width=4).grid(row=2,column=2)
13          B(m,text=7,command=lambda:e.insert(END,7),width=4).grid(row=3,column=0)
14          B(m,text=8,command=lambda:e.insert(END,8),width=4).grid(row=3,column=1)
15          B(m,text=9,command=lambda:e.insert(END,9),width=4).grid(row=3,column=2)
16          B(m,text="+",command=lambda:e.insert(END,"+"),width=4).grid(row=4,column=0)
17          B(m,text="-",command=lambda:e.insert(END,"-"),width=4).grid(row=4,column=1)
18          B(m,text="/",command=lambda:e.insert(END,"/"),width=4).grid(row=4,column=2)
19          B(m,text="*",command=lambda:e.insert(END,"*"),width=4).grid(row=5,column=0)
20          B(m,text="clr",command=lambda:e.delete(0,END),width=4).grid(row=5,column=1)
21          B(m,text="=",command=self.computendresult,width=4).grid(row=5,column=2)
22          self.ent=e
23      def computendresult(self):
24          er=eval(self.ent.get())
25          self.ent.delete(0,END)
26          self.ent.insert(END,er)
27
28  root=Tk()
29  app=App(root)
30  root.mainloop()
```

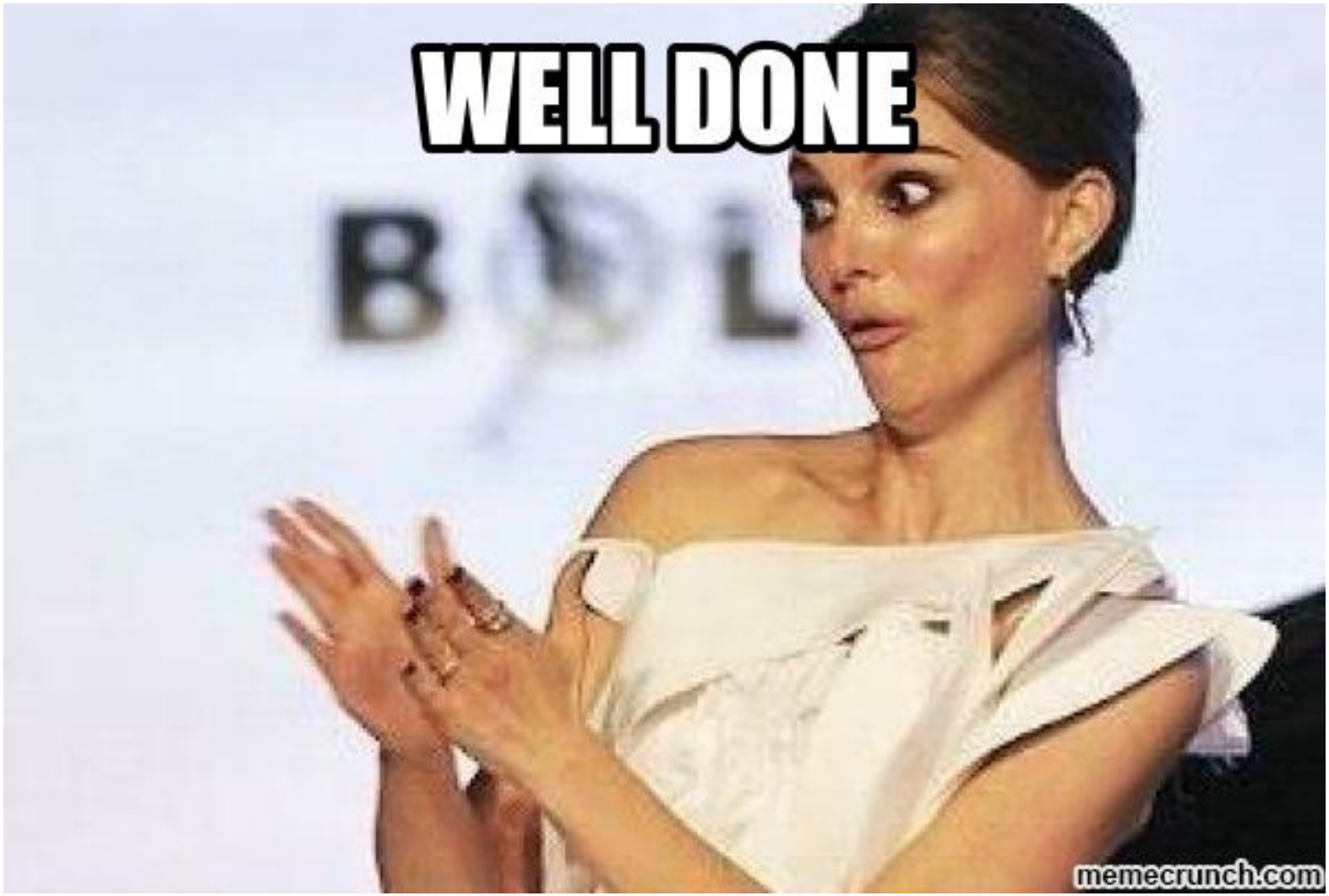
Calculator in action



LAST MINI TASK

Convert the calculator to a Mac-app and proudly sent it to someone :)

FINALLY, THE LADIES ARE IMPRESSED :)



SOME NOTES ON THE FINAL EXAM

- the exam will be oral in the computer room
- bring all your tasks; preferentially as print and send me the last one as email: rokofler 'at' gmail.com
- Note: I prefer a self-made script with some problems over any copied script that works
- Bring your student ID (Studentenausweis)
- Part 1: run a Python script from the command line, including parameters and redirect
- Part 2: I pick a script and you need to explain it to me. Than I ask stupid questions, like what happens if I delete this line of code
- Part 3: another script or your last task if well done
- Register for exam at http://drrobertkofler.wikispaces.com/PythonLecture