Comprehension

Thomas Schwarz, SJ

Contents

- The random module
- Some repetition and new stuff about loops
- List and dictionary comprehension

The random module

Python
Marquette University

A Monte Carlo Method for Area calculation

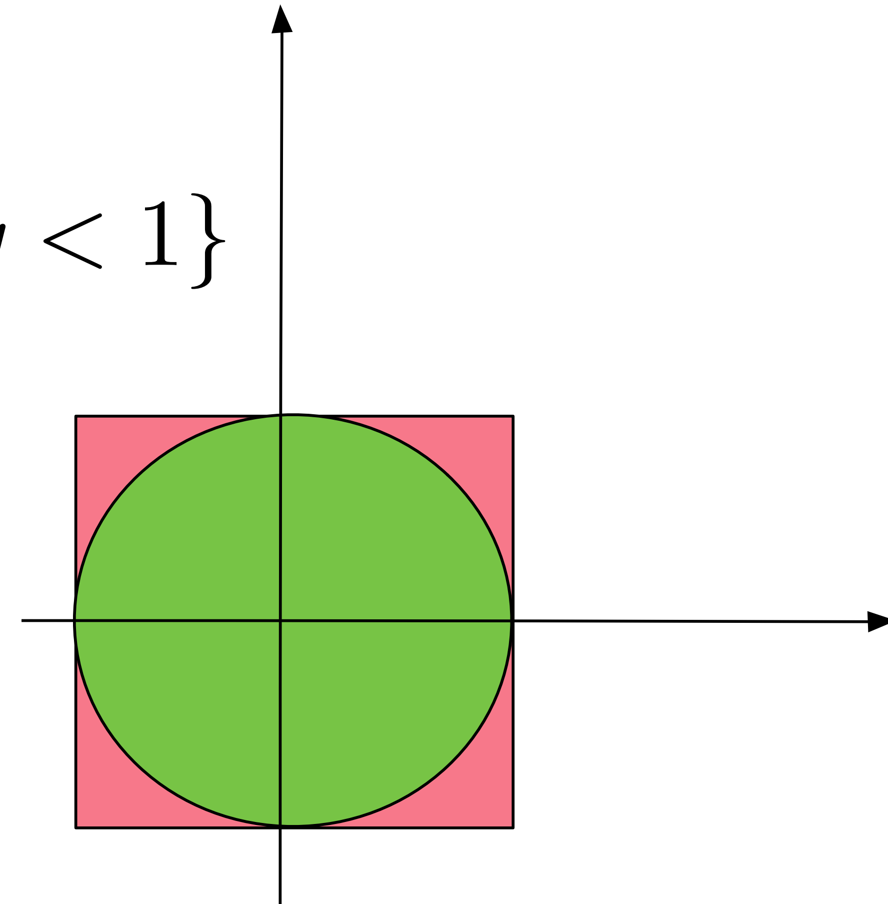
- Calculate the area of a circle of radius 1 $A = r^2 \cdot \pi$
  - Can be done analytically:
  - Can be done with <u>Monte Carlo Method</u>
    - Use pseudo-random numbers in order to determine values probabilistically
    - Named after Stanislav Ulam
      - Used for work on the thermo-nuclear device

A Monte Carlo Method for Area calculation

$$\{(x, y) | x^2 + y^2 < 1\}$$

$$\{(x, y) | -1 < x < 1, -1 < y < 1\}$$

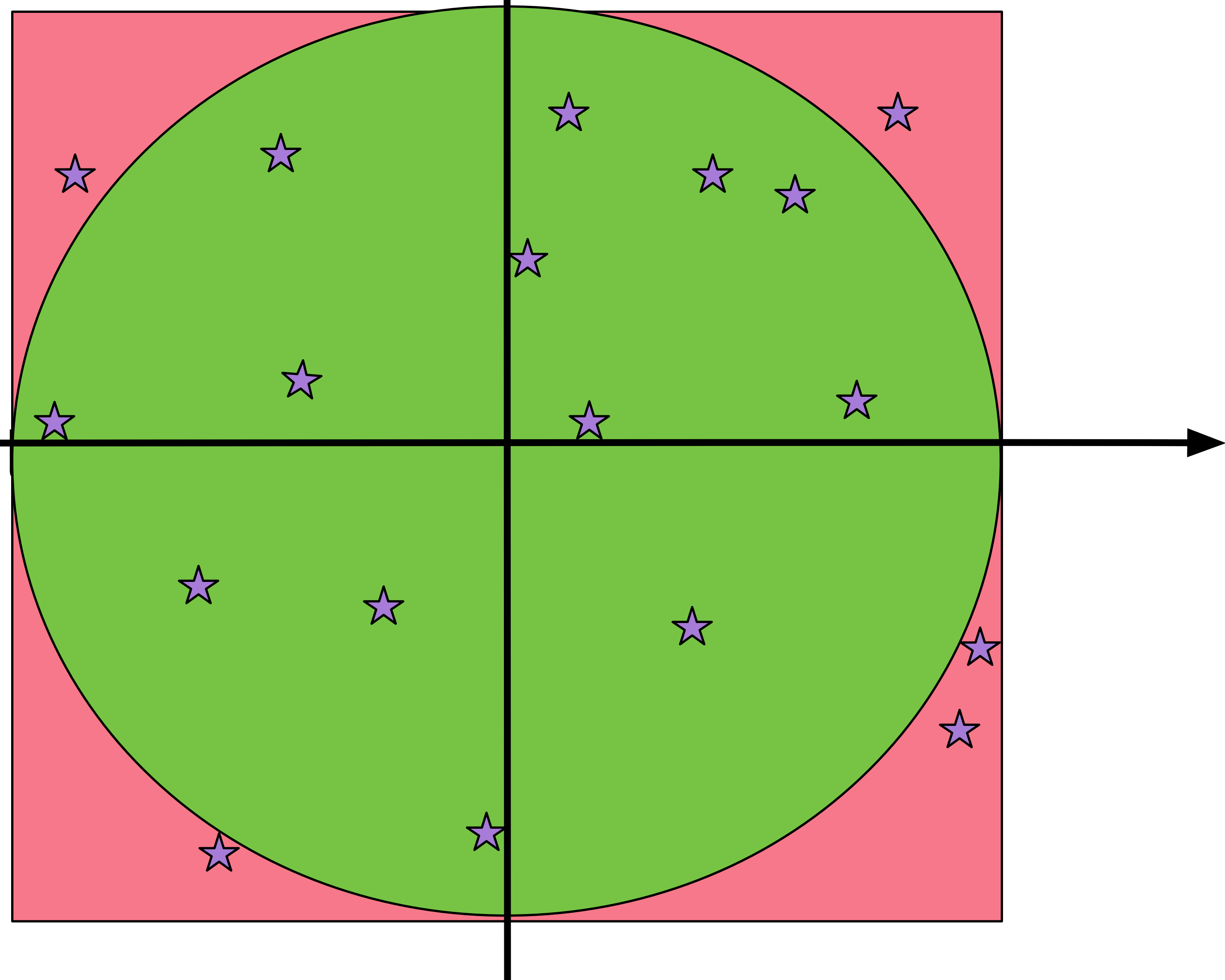
- Inscribe Circle with a square
- Circle:
- Square:



A Monte Carlo Method for Area calculation

- Method:
  - Choose n random points
  - m points inside circe

$$\frac{\text{Area of Circle}}{\text{Area of Square}} \approx \frac{m}{n}$$



Random Number Generation

- Computers are deterministic (one hopes) and using a deterministic device to generate randomness is not possible
  - Modern systems can use physical phenomena
    - Geiger counters for radioactive materials
    - Atmospheric radio noise
  - But for large sets of seemingly random numbers, use pseudo-random number generators
    - Create deterministically based on a seemingly random seed output that passes statistical tests for randomness

Random Number Generation in Python

- Sophisticated methods to generate seemingly random sequences of numbers
- Part of a <u>module</u> called random

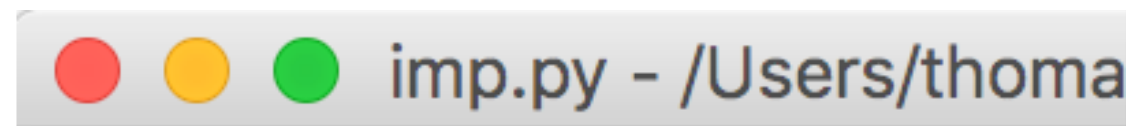
Interlude: Python Modules

- Anyone can create a python module
  - Just a file with extension .py
  - In a directory in the Python path, which is set for the OS
  - Or just in the same directory as files that use the module
- A module contains definitions of variables and functions
  - Any python script that imports the module can use them

Interlude: Python Modules

- Predefined modules
  - Python defines many modules
    - We already have seen math and os
- To use such a module, say
  - `import random`
    - in order to use the functions within random

Interlude: Python Modules



- If I just import ... an use its functions by p
  -

```
import random

for _ in range(10):
    print(random.random())
```

Using the function random inside the module random

Interlude: Python Modules

- If I want to av[...]ne I can use an "as" clause t[...]ne module within the script

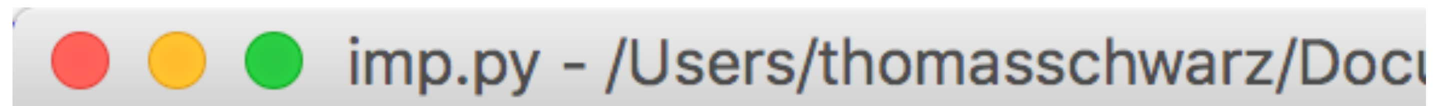
imp.py - /Users/tho

```
import random as rd

for _ in range(10):
    print(rd.random())
```

```
Using the same function in the same module,
but now after internally renaming the module
```

Interlude: Python Modules




```
from random import uniform, randint

for _ in range(10):
    print(uniform(0,2), randint(0,10))
```

- By using … variables and funct … me

Importing the two functions uniform and randint from the random module.

Interlude: Python Modules

- I could even ... ule
- his ca ... nction with the nam

imp.py - /Users/thomasschwarz/Do

```
from random import *

for _ in range(10):
    print(uniform(0,2), randint(0,10))
```

A dangerous practice: Importing all functions from a module

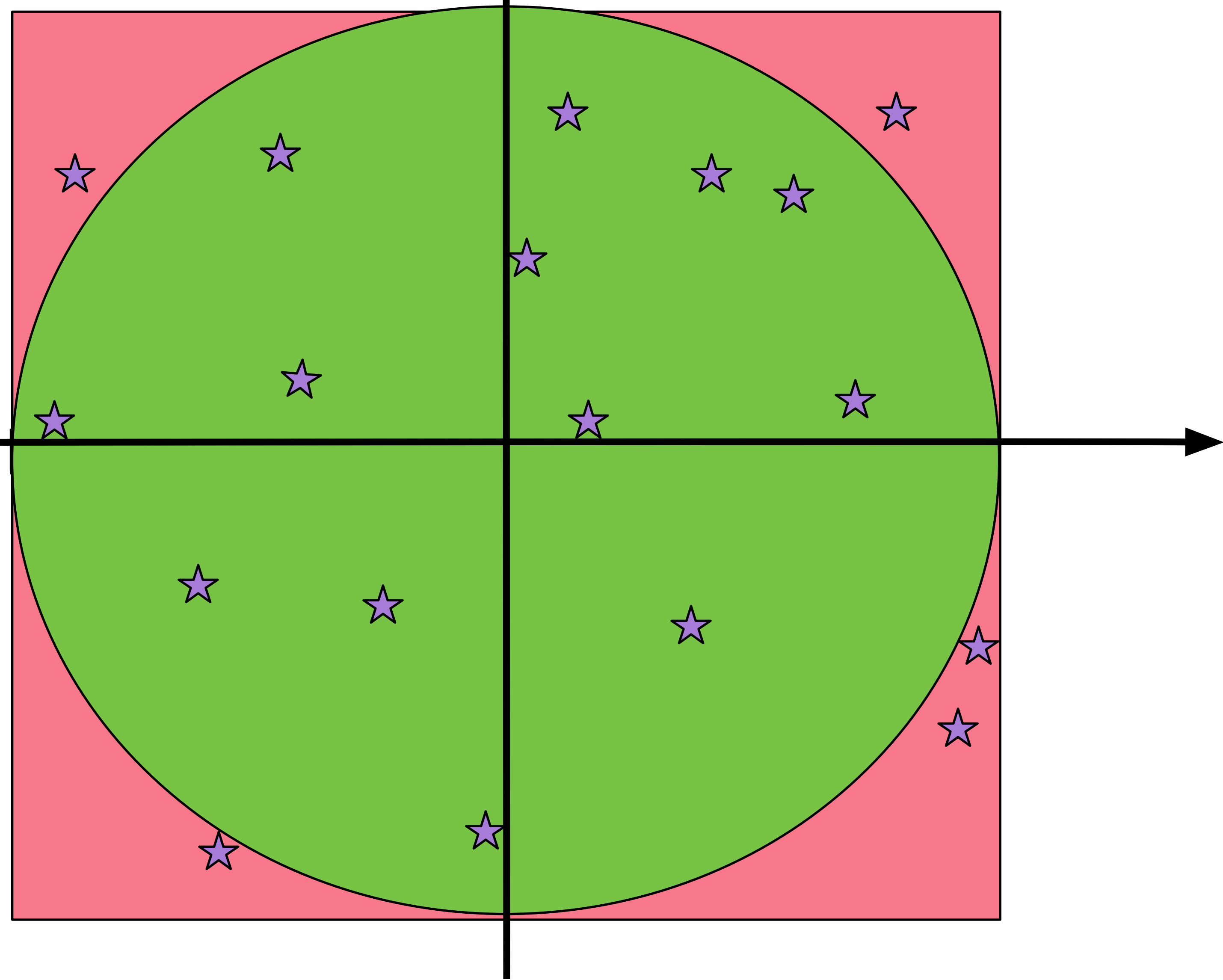
Random Module

- Important functions in the random module
  - `random.randint(a, b)` Selects a random integer between a and b (boundaries included)
  - random.uniform(a, b) Selects a random float between a and b
  - random.random( ) Selects a random number between 0 and 1

A Monte Carlo Method for Area calculation

- Method:
  - Choose n random poin
  - m points inside circe

$$\frac{\text{Area of Circle}}{\text{Area of Square}} \approx \frac{m}{n}$$



A Monte Carlo Method for Area calculation

- Use random module
  - `random.uniform(-1,1)` generates random number between -1 and 1
  - Generating *20* random numbers:

```
import random

for i in range(20):
    x = random.uniform(-1,1)
    y = random.uniform(-1,1)
    print("({:6.3f},{:6.3f})".format(x,y))
```

A Monte Carlo Method for Area calculation

```
import random

def approx(N):
    count = 0
    for i in range(N):
        x = random.uniform(-1,1)
        y = random.uniform(-1,1)
        if x*x+y*y<1:
            count += 1
    return (4*count/N)
```

- We then only count those that are inside the circle

A Monte Carlo Method for Area Calculations

$$\frac{\text{count}}{N} \approx \frac{\text{Area Circle}}{\text{Area Box}}$$

```
import random

def approx(N):
    count = 0
    for i in range(N):
        x = random.uniform(-1,1)
        y = random.uniform(-1,1)
        if x*x+y*y<1:
            count += 1
    return (4*count/N)
```

- Since and the area of the box is 4
- we return $\frac{4\text{count}}{N}$

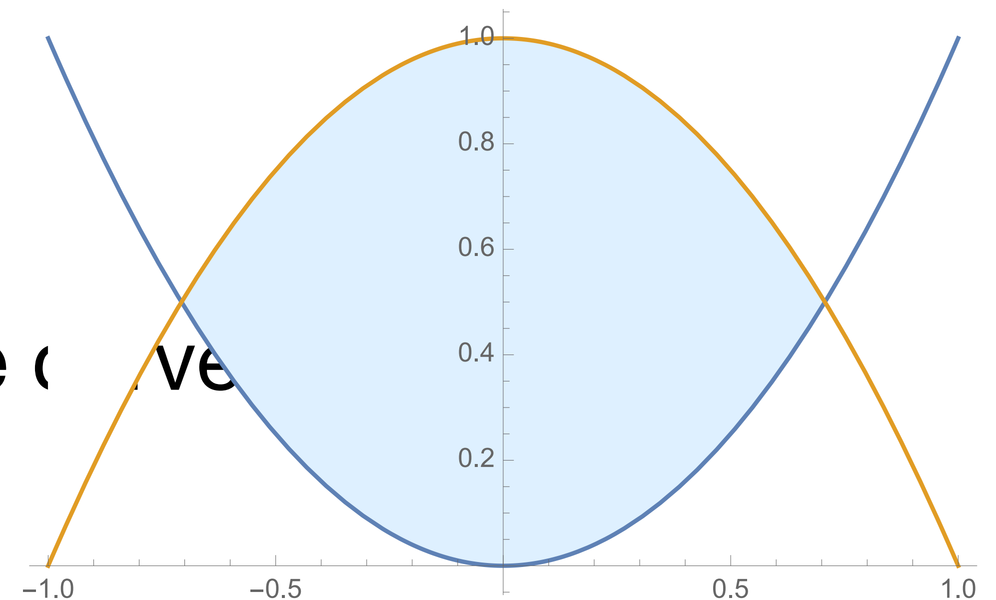
A Monte Carlo Method for Area calculation

- Need few random point to get a general idea
- Need lots to get any good accuracy
- Method of choice used to determine 6-dimensional integrals for simulation of quantum decay where accuracy is not as important as speed

A Monte Carlo Method for Area calculation

- Your task:
  - Determine the area between the curves

$$y = x^2$$
$$y = 1 - x^2$$



- Hint: We draw points in the rectangle [-1,1] x [0,1]
- (x,y) lies in the area if

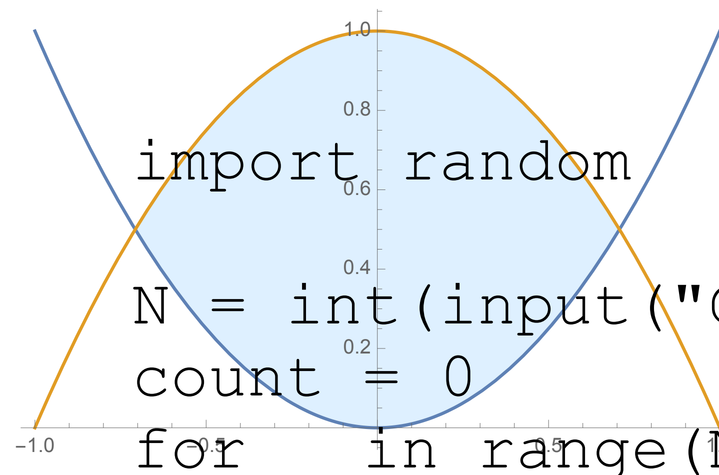
$$x^2 < y < 1 - x^2$$

A Monte Carlo Method for Area calculation

Select random points in the box [-1,1] x [0,1]

Count the number of times that the point falls in the area

Multiply the ratio count / #pts by the area of the box, which is 2




```
import random

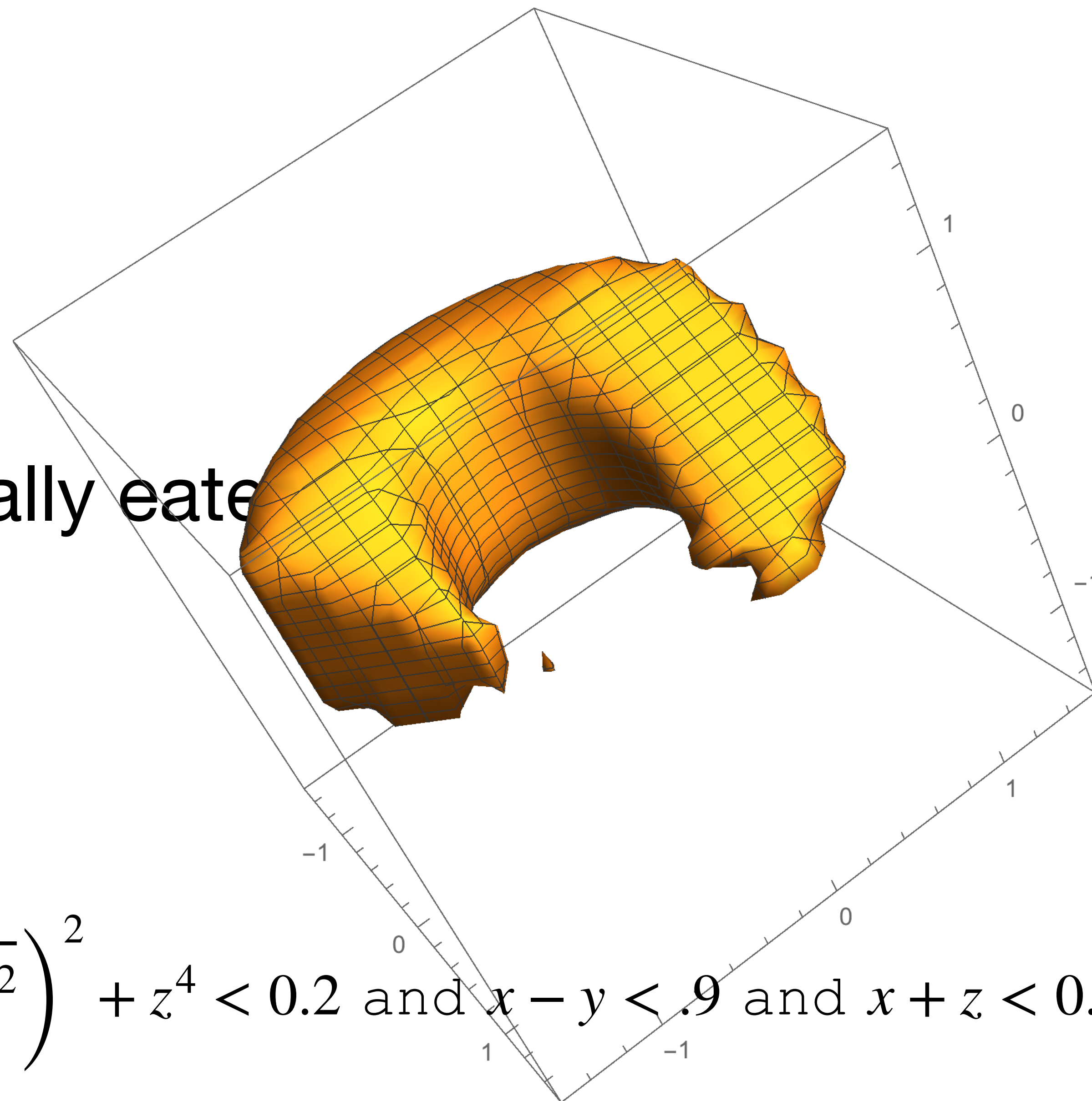
N = int(input("Give the number of random points: "))
count = 0
for _ in range(N):
    x = random.uniform(-1,1)
    y = random.uniform(0,1)
    if x*x < y < 1-x*x:
        count += 1
print("The area is approximately", count*2/N)
```

Monte-Carlo Volume Calculation

- Sometimes, Monte-Carlo is the method of choice
  - When there is no need for super-precision
  - When the volume is not easily evaluated using analytic methods.

Volume Calculation

- A partially eate

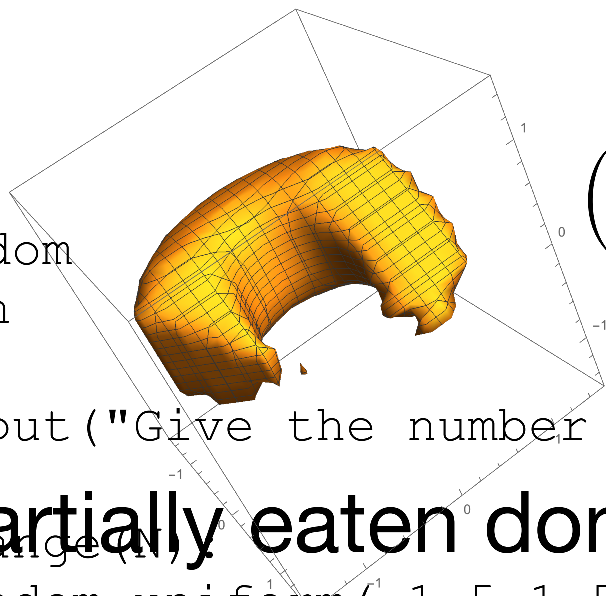


$$\left(1-\sqrt{x^2+y^2}\right)^2 + z^4 < 0.2 \texttt{ and } x - y < .9 \texttt{ and } x + z < 0.1 \texttt{ and } x + y < 1.8$$

Volume Calculation

- Monte Carlo:
  - Select random points in the box $-1.5<x<1.5$, $-1.5<y<1.5$, $-1.5<z<1.5$.
  - Check whether they are inside the donut
  - Count over total number is approximately area of donut over area of box (which is 9).

Volume Calculation



$$\left(1-\sqrt{x^2+y^2}\right)^2+z^4<0.2 \text{ and } x-y<.9 \text{ and } x+z<0.1 \text{ and } x+y<1.8$$

- A partially eaten donut

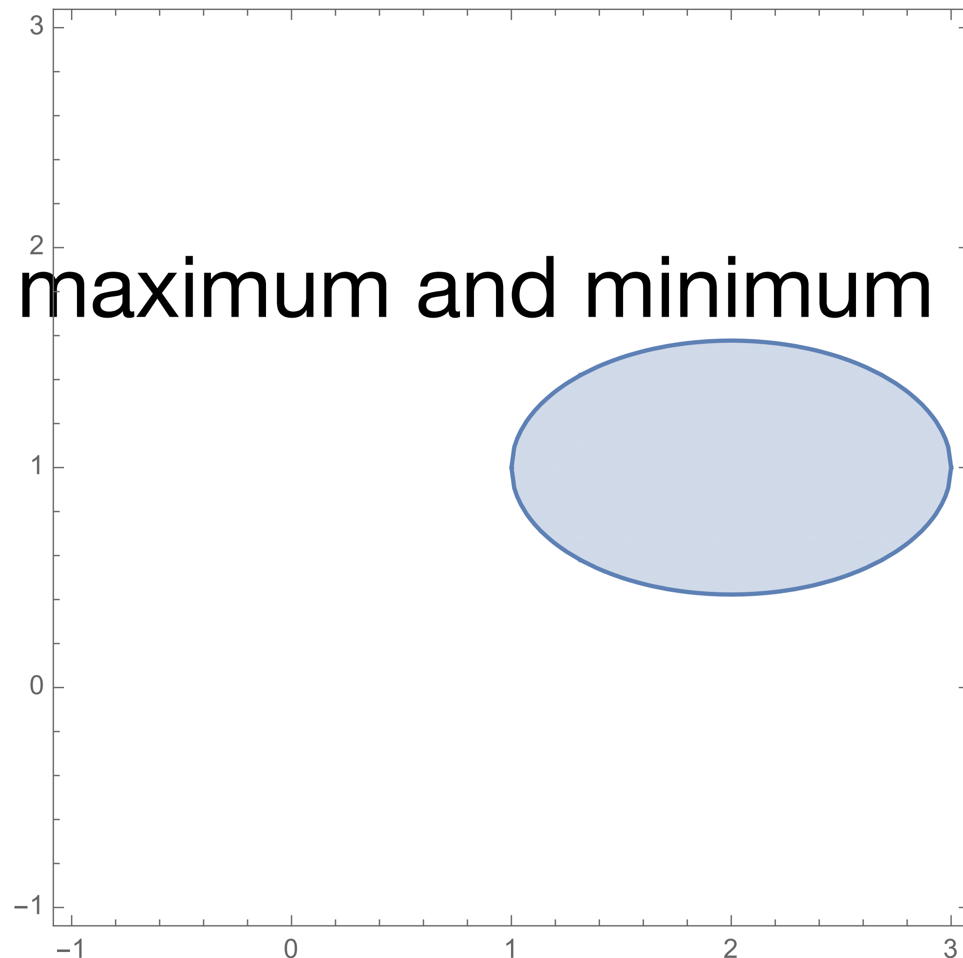
```
import random
import math

N = int(input("Give the number of random points: "))
count = 0
for _ in range(N):
    x = random.uniform(-1.5,1.5)
    y = random.uniform(-1.5,1.5)
    z = random.uniform(-1.5,1.5)
    if (1-math.sqrt(x**2+y**2))**2+z**4<0.2 and x-y<0.9 and x+z<0.1 and x+y<1.8:
        count += 1
print("The area is approximately", count*27/N)
```

Additional Exercises

$$\{(x,y)|(x-2)^2+3*(y-1)^2<1\}$$

- Find the area of
    - Hint: First determine maximum and minimum values for x and y



Loops

Loops

- Python does not use indices in for loops
    - ```
      animals = ['bird', 'dog', 'cat']

      for animal in animals:
          print(f'I have a {animal})
      ```
      ```

Loops

- If we need indices in Python, we can use enumerate

```
colors = ['blue', 'yellow', 'red', 'green']

for i, color in enumerate(colors):
    print(i, color)
```

- `enumerate` returns an enumerate object which is an iterator that allows us to use a for loop
- You can even change the "index"

```
for i, color in enumerate(colors, start = 1):
```

Loops

- Dictionaries:
  - The for loop takes the keys

```
animals = {'dog' : 3, 'cat' : 5, 'bird' : 1, 'hamster': 2}

for animal in animals:
    print(f'I have {animals[animal]} {animal}(s)')
```

Loops

- Dictionaries:
  - We can avoid the bracket notation using items
  - items returns an items object, but that is not important to us

```
animals = {'dog' : 3, 'cat' : 5, 'bird' : 1, 'hamster': 2}

for animal, count in animals.items():
    print(f'I have {count} {animal}(s)')
```

Comprehension

Thomas Schwarz, SJ
Marquette University

Programming Styles

- Styles of Programming
  - Imperative Programming:
    - Describe in detail how computation proceeds
    - Basically, change states of variables
    - This is what we practiced up till now

Programming Styles

- Functional Programming
  - Define functions
    - Specify program behavior by executing nested functions
    - Pure functional programming: No variables that capture a state
  - Advantage: Easier to prove programming correctness

Programming Styles

- Declarative Programming
  - Specify what a program should do
    - System figures out how to do it.
    - Example 1: Prolog (Classic AI programming language)
      - Specify rules in Prolog:
        - `animal(X) :- cat(X)` means every cat is an animal
        - `?- cat(tom).` means that tom is a cat
      - You can ask about the world defined by these rules
        - `?- animal(X).` asks for what things are animals
      - Prolog consists of rules and base facts, then on its own finds out other facts.

Programming Styles

- Declarative Programming:
    - Example 2: SQL — Database Language
        - Database consists of relations stored in various tables

Marquette_ID	First_Name	Family_Name	Address
123123007	David	Roy	1984 31st Street, Milwaukee, WI 54321
97007007	Thomas	Schwarz	4821 Wisconsin Ave, Milwaukee, WI 54213
14309873	Joseph	Cuelho	9821 12th Avenue, Milwaukee, WI 54321
90874132	Donald	Drumpf	321 Pennsylvania Ave, Madison, WI 32451

Programming Styles

- Declarative Programming:
    - Example SQL:

```
SELECT first_name, family_name FROM
addresses, classes
WHERE classes.name = “COSC1010” and
classes.role = “instructor” and
classes.id = addresses.id
```

- SQL statement describes all combinations of record pieces

Programming Styles

- Declarative Programming:
  - Example SQL:
    - SQL statement describes all combinations of record pieces
    - How the database engine performs the query is **not** specified
    - In fact, for complicated queries, the database will try out several ways before selecting the actual algorithms

Programming Styles

- Object-Oriented Programming
  - Program defined various objects
    - Objects have data and methods
      - E.g. Marquette Persons have IDs, names, addresses, …
      - Classes have lists of participants
  - We will learn Object-Oriented (OO) programming in this class

Comprehension

- List comprehension is used in functional programming but it becomes handy
  - We define a list with a for clause within the brackets that define the list.
  - Here are two ways to construct a list consisting of squares

```
lista = []
for i in range(100):
    lista.append(i**2)
```

```
lista = [i**2 for i in range(100)]
```

Comprehension

```
[x**2 for x in range(100)]
```

output expression

generator expression

variable

list
(or list-like expression)

Self Test

- The following code fragment defines a list of elements
- Use list comprehension in order to generate the same list

```
>>> lista = []
>>> for i in range(10):
        lista.append(i**3-i**2+i-1)


>>> lista
[-1, 0, 5, 20, 51, 104, 185, 300, 455, 656]
```

Self Test

Pause the presentation until you have solved the problem

Self Test Solution

```
>>> lista = [i**3-i**2+i-1 for i in range(10)]
>>> lista
[-1, 0, 5, 20, 51, 104, 185, 300, 455, 656]
```

Comprehension

```
[ x**2 for x in range(100) if x%2 == 0 ]
```

- List comprehension can add an if-condition

  - Result is now all even squares.

Comprehension

- List comprehension can be quite involved
  - Remember that we can check for types of variables

    `isinstance(345, int)`
  - We use the built-in function `isinstance( )`
  - Example: is True
  - Application to list comprehension: Squaring the

```
>>> a_list = [1, "4", 9, "a", 0, 4]
>>> [e**2 for e in a_list if isinstance(e, int)]
[1, 81, 0, 16]
```

Comprehension

- We can nest comprehensions
- A list of all composite numbers between 2 and 100.
  - A composite number is a product of two integers *i* and *j* that are larger than 1.

```
[i*j for i in range(2,51) for j in range(2,101) if i*j < 100]
```

- However, the result contains many repeated numbers

```
[4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66,
68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51,
54, 57, 60, 63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93, 96, 99, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68,
72, 76, 80, 84, 88, 92, 96, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 12, 18, 24, 30, 36, 42,
48, 54, 60, 66, 72, 78, 84, 90, 96, 14, 21, 28, 35, 42, 49, 56, 63, 70, 77, 84, 91, 98, 16, 24, 32, 40, 48, 56, 64, 72, 80,
88, 96, 18, 27, 36, 45, 54, 63, 72, 81, 90, 99, 20, 30, 40, 50, 60, 70, 80, 90, 22, 33, 44, 55, 66, 77, 88, 99, 24, 36, 48,
60, 72, 84, 96, 26, 39, 52, 65, 78, 91, 28, 42, 56, 70, 84, 98, 30, 45, 60, 75, 90, 32, 48, 64, 80, 96, 34, 51, 68, 85, 36,
54, 72, 90, 38, 57, 76, 95, 40, 60, 80, 42, 63, 84, 44, 66, 88, 46, 69, 92, 48, 72, 96, 50, 75, 52, 78, 54, 81, 56, 84, 58,
87, 60, 90, 62, 93, 64, 96, 66, 99, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98]
```

Comprehensions

- Luckily, we can use a set instead:

```
{i*j for i in range(2,51) for j in range(2,51) if i*j < 100}
```

- The difference is just curly brackets instead of rectangular brackets
- The result is now simpler:

```
{4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 22, 24, 25,
26, 27, 28, 30, 32, 33, 34, 35, 36, 38, 39, 40, 42, 44,
45, 46, 48, 49, 50, 51, 52, 54, 55, 56, 57, 58, 60, 62,
63, 64, 65, 66, 68, 69, 70, 72, 74, 75, 76, 77, 78, 80,
81, 82, 84, 85, 86, 87, 88, 90, 91, 92, 93, 94, 95, 96,
98, 99}
```

Comprehensions

- We can now get all of the prime numbers between 2 and 100 by using this set, using comprehension on top of comprehension

```
{i for i in range(2,100) if i not in
{i*j for i in range(2,51) for j in range(2,51) if i*j < 100}}
```

- This is cool but will not win any price for clarity
- You can make it more comprehensible if you define a set of composite numbers before using it

Self Test

- Use the previous example to generate a set of all numbers between 1 and 100 (included) that are **not** squares

Self Test Solution

```
seta = {i for i in range(1,101) if i not in {i*i for i in range(10)}}
```

Comprehensions

- You can also use comprehension on dictionaries
- Here is how you create a dictionary that associates

```
>>> {i*i: i for i in range(101)}
{0: 0, 1: 1, 4: 2, 9: 3, 16: 4, 25: 5, 36: 6, 49: 7, 64: 8, 81: 9, 100: 10, 121:
 11, 144: 12, 169: 13, 196: 14, 225: 15, 256: 16, 289: 17, 324: 18, 361: 19, 400
: 20, 441: 21, 484: 22, 529: 23, 576: 24, 625: 25, 676: 26, 729: 27, 784: 28, 84
1: 29, 900: 30, 961: 31, 1024: 32, 1089: 33, 1156: 34, 1225: 35, 1296: 36, 1369:
 37, 1444: 38, 1521: 39, 1600: 40, 1681: 41, 1764: 42, 1849: 43, 1936: 44, 2025:
 45, 2116: 46, 2209: 47, 2304: 48, 2401: 49, 2500: 50, 2601: 51, 2704: 52, 2809:
 53, 2916: 54, 3025: 55, 3136: 56, 3249: 57, 3364: 58, 3481: 59, 3600: 60, 3721:
 61, 3844: 62, 3969: 63, 4096: 64, 4225: 65, 4356: 66, 4489: 67, 4624: 68, 4761:
 69, 4900: 70, 5041: 71, 5184: 72, 5329: 73, 5476: 74, 5625: 75, 5776: 76, 5929:
 77, 6084: 78, 6241: 79, 6400: 80, 6561: 81, 6724: 82, 6889: 83, 7056: 84, 7225:
 85, 7396: 86, 7569: 87, 7744: 88, 7921: 89, 8100: 90, 8281: 91, 8464: 92, 8649:
 93, 8836: 94, 9025: 95, 9216: 96, 9409: 97, 9604: 98, 9801: 99, 10000: 100}
```

Comprehensions

- And here is how you can try to "invert" a dictionary where the roles of keys and values are swapped

```
dnew = {d[key]:key for key in d}
```

- This one works well. because the values are different for different key

```
>>> d = {1:4, 2:5, 3:7, 4:8, 5:9}
>>> {d[key]:key for key in d}
{4: 1, 5: 2, 7: 3, 8: 4, 9: 5}
```

- And thi

```
>>> d = {1:4, 2:5, 3:4, 4:5, 6:7, 7:6}
>>> {d[key]:key for key in d}
{4: 3, 5: 4, 7: 6, 6: 7}
```

Self Test

- You are given a function `func` that takes one integer argument
- You want to create a memoization dictionary that associates `i` for `i` in `range(100)` with `func(i)`

Self Test Answer

```
mem_func = {i: func(i) for i in range(101)}
```

```
func = lambda x: 3*x+4
```

gives

```
>>> func = lambda x: 3*x+4
>>> mem = {x: func(x) for x in range(101)}
>>> mem
{0: 4, 1: 7, 2: 10, 3: 13, 4: 16, 5: 19, 6: 22, 7: 25, 8: 28, 9: 31, 10: 34, 11:
 37, 12: 40, 13: 43, 14: 46, 15: 49, 16: 52, 17: 55, 18: 58, 19: 61, 20: 64, 21:
 67, 22: 70, 23: 73, 24: 76, 25: 79, 26: 82, 27: 85, 28: 88, 29: 91, 30: 94, 31:
 97, 32: 100, 33: 103, 34: 106, 35: 109, 36: 112, 37: 115, 38: 118, 39: 121, 40:
 124, 41: 127, 42: 130, 43: 133, 44: 136, 45: 139, 46: 142, 47: 145, 48: 148, 49
: 151, 50: 154, 51: 157, 52: 160, 53: 163, 54: 166, 55: 169, 56: 172, 57: 175, 5
8: 178, 59: 181, 60: 184, 61: 187, 62: 190, 63: 193, 64: 196, 65: 199, 66: 202,
67: 205, 68: 208, 69: 211, 70: 214, 71: 217, 72: 220, 73: 223, 74: 226, 75: 229,
 76: 232, 77: 235, 78: 238, 79: 241, 80: 244, 81: 247, 82: 250, 83: 253, 84: 256
, 85: 259, 86: 262, 87: 265, 88: 268, 89: 271, 90: 274, 91: 277, 92: 280, 93: 28
3, 94: 286, 95: 289, 96: 292, 97: 295, 98: 298, 99: 301, 100: 304}
```

Map, Filter

Map

- Map allows you to apply a function to all elements of a list
- Example:

```
func = lambda x: x+3
list(map(func, [2,3,4]))
```

- Why the list? map returns an iterator (so that it does not waste m

```
>>> func = lambda x: x+3
>>> list(map(func, [2,3,4]))
[5, 6, 7]
```

Filter

- You filter a list by applying a condition
- The result is the list formed by all elements that satisfy the condition
  - You need to have a boolean function, i.e. a function that returns True or False
  - Here is an example of such a function:

```
lambda x: x%2==0
```

    - Returns True if x is divisible by 2
    - Returns False otherwise
    - x%2 is zero if and only if x is even

Filter

- The function `filter(function, sequence)` return an

```
>>> fibonacci = [0, 1, 1, 2, 3, 5, 8, 13, 21, 44, 65, 109, 174, 283]
>>> list(filter(lambda x: x%2==0, fibonacci))
[0, 2, 8, 44, 174]
>>> list(filter(lambda x: x%2==1, fibonacci))
[1, 1, 3, 5, 13, 21, 65, 109, 283]
>>> 
```

Comprehension in Action

Python

Getting the listing of a directory

- Task: Generate a listing of all files in a directory that end in ".py"

```
[filename for filename in os.listdir(directoryname)
     if filename.endswith(".py")]
```

- Tool: import the os module and use listdir

Creating sub-directories

- Task: We want to create a sub-dictionary of a dictionary where the keys are restricted by a condition
  - Use dictionary comprehension

```
def evenkeys(dictionary):
    return { i:dictionary[i] for i in dictionary if i%2==0}
```

Filtering a list

- We want to filter a list using a criterion

  1. We

  2. We

- Ex

```
>>> rlist
[20, -1, 3, 0, 17, 1, 20, 19, 24, 4, 21, 0, 4, 7, 20, 2, 1, 13, 0, 21, 23, 6, 2,
 22, 4, 3, 6, 2, 13, -5, 3, 13, 20, 23, 14, 13, 13, 20, 10, 24, 9, -1, -4, 22, 1
5, 21, 18, -1, 16, 13, 1, 3, 12, 21, 0, 9, 4, 24, -3, 4, 10, 8, 1, 19, 3, 20, 4,
 5, 25, 8, 8, 14, -5, 23, 24, 14, 1, 0, -5, -3, 3, -4, 11, 1, 8, 17, 2, 2, 23, 6
, 2, 25, 15, 4, 23, 20, 5, -3, 11, 16]
>>> list(filter(lambda x: x>0, rlist))
[20, 3, 17, 1, 20, 19, 24, 4, 21, 4, 7, 20, 2, 1, 13, 21, 23, 6, 2, 22, 4, 3, 6,
 2, 13, 3, 13, 20, 23, 14, 13, 13, 20, 10, 24, 9, 22, 15, 21, 18, 16, 13, 1, 3,
12, 21, 9, 4, 24, 4, 10, 8, 1, 19, 3, 20, 4, 5, 25, 8, 8, 14, 23, 24, 14, 1, 3,
11, 1, 8, 17, 2, 2, 23, 6, 2, 25, 15, 4, 23, 20, 5, 11, 16]
>>> [x for x in rlist if x>0]
[20, 3, 17, 1, 20, 19, 24, 4, 21, 4, 7, 20, 2, 1, 13, 21, 23, 6, 2, 22, 4, 3, 6,
 2, 13, 3, 13, 20, 23, 14, 13, 13, 20, 10, 24, 9, 22, 15, 21, 18, 16, 13, 1, 3,
12, 21, 9, 4, 24, 4, 10, 8, 1, 19, 3, 20, 4, 5, 25, 8, 8, 14, 23, 24, 14, 1, 3,
11, 1, 8, 17, 2, 2, 23, 6, 2, 25, 15, 4, 23, 20, 5, 11, 16]
```

Mapping a list

```
>>> rlist =[random.randint(-10,20) for _ in range(20)]
>>> rlist
[-2, -9, 20, -10, -9, 19, -4, 1, 16, 3, 8, -10, 4, -2, 11, 8, 11, -7, -2, -3]
>>> list(map(lambda x: (x-6)**2, rlist))
[64, 225, 196, 256, 225, 169, 100, 25, 100, 9, 4, 256, 4, 64, 25, 4, 25, 169, 64
, 81]
>>> [(x-6)**2 for x in rlist]
[64, 225, 196, 256, 225, 169, 100, 25, 100, 9, 4, 256, 4, 64, 25, 4, 25, 169, 64
, 81]
```

Zip

Zip

- Often we have related data in a number of lists
  - Example: list of student names, list of grades, list of high school
    - [“Frankieboy”, “Violet”, “Kumar”, “Dshenghis”]
    - [“D”, “A”, “B”, “C”]
    - [“MPS1”, “MH”, “MH”, “MPS59”]
  - Zipping will create a zip object that generates the tuples (“Frankieboy”, “D”, “MPS1”), (“Violet”,“A”,”MH”), (“Kumar”, “B”, “MH”), (“Dshenghis”,”C”, “MPS59”)

Zip

```
>>> names = ["Albertina", "Bertram", "Chris", "David"]
>>> grades = ["A", "B", "C", "D"]
>>> highschools = ["MH", "SHH", "LGH", "MHT"]
>>> zip(names, grades, highschools)
<zip object at 0x1153e8bc8>
>>> list(zip(names, grades, highschools))
[('Albertina', 'A', 'MH'), ('Bertram', 'B', 'SHH'), ('Chris', 'C', 'LGH'), ('David', 'D', 'MHT')]
>>> [ (names[i], grades[i], highschools[i]) for i in range(len(names))]
[('Albertina', 'A', 'MH'), ('Bertram', 'B', 'SHH'), ('Chris', 'C', 'LGH'), ('David', 'D', 'MHT')]
```

Zip

- What happens if you give zip iterables of different length
    - E.g. a list of 5, a list of 4 and a list of 3 elements?
    - The result is a zip object of length the minimum of the lengths.

Zip

- Un
  - I ... an
    k

```
>>> names = ["Albertina", "Bertram", "Chris", "David"]
>>> grades = ["A", "B", "C", "D"]
>>> highschools = ["MH", "SHH", "MPS57", "LGH"]
>>> alist = list(zip(names, grades, highschools))
>>> alist
[('Albertina', 'A', 'MH'), ('Bertram', 'B', 'SHH'), ('Chris', 'C', 'MPS57'), ('D
avid', 'D', 'LGH')]
>>> list(zip(*alist))
[('Albertina', 'Bertram', 'Chris', 'David'), ('A', 'B', 'C', 'D'), ('MH', 'SHH',
 'MPS57', 'LGH')]
>>> names, grades, highschools = tuple(list(zip(*alist)))
>>> names
('Albertina', 'Bertram', 'Chris', 'David')
>>> grades
('A', 'B', 'C', 'D')
>>> highschools
('MH', 'SHH', 'MPS57', 'LGH')
```

Exercises

Exercise

- Use list comprehension:
  - Flatten a matrix
    - Example: [[1, 2, 3], [4, 5, 6], [7, 8, 9]] —> [1,2,3,4,5,6,7,8,9]

Solution

- Loop Solution:
  - Using extend

```
def flatten1(matrix):
    result = []
    for row in matrix:
        result.extend(row)
    return result
```

  - But this cannot be translated

Solution

- A loop solution that can be translated

```
def flatten2(matrix):
    result = []
    for row in matrix:
        for item in row:
            result.append(item)
    return result
```

- This is not english, but Python: for row in matrix for item in row

Solution

- Now we can do comprehension with the same order of for loops

```
def flatten3(matrix):
    return [ item for row in matrix for item in row]
```

Exercise

- Given a list, subtract its reverse from itself
    - [10, 7, 5, 4, 2, 1] —> [10-1, 7-2, 5-4, 4-5, 2-7, 1-10]

Solution

- Loop version:
  - Use a slice to get the reverse of the list

```
def clw(lista):
    result = []
    for first, second in zip(lista, lista[::-1]):
        result.append(first-second)
    return result
```

Solution

- Translated into comprehension

```
def clwc(lista):
    return [first - sec for first, sec in zip(lista, lista[::-1])]
```

Exercise

- Given a matrix, calculate its negative

```
[  [1,2,4],
   [2,5,8],
   [3,3,3],
   [5,4,2]
]
```

Solution

- A double loop

```
def neg1(matrix):
    result = []
    for row in matrix:
        new_row = [ ]
        for item in row:
            new_row.append(-item)
        result.append(new_row)
    return result
```

Solution

- A single loop with one interior comprehension

```
def neg2(matrix):
    result = []
    for row in matrix:
        result.append([-item for item in row])
    return result
```

Solution

- A double comprehension (which shows that you might not want to overdo comprehension)

```
def neg3(matrix):
    return [ [-item for item in row] for row in matrix]
```

Generator Comprehension

- We can use comprehension on generators
- Called **generator expressions**
  - Generators are defined with round parentheses

```
squares = (n**2 for n in range(1, 100))
```

```
>>> next(squares)
1
>>> next(squares)
4
>>> next(squares)
9
>>> next(squares)
16
>>> next(squares)
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    next(squares)
StopIteration
```

Generator Comprehension

- The generator expression can be called with next( )
  - However, what if we want an infinite generator?
    - Could define a generator the old fashioned way

```
def squares1():
    n=0
    while True:
        n+=1
        yield(n**2)
```

```
>>> a = squares1()
>>> while(True):
        print(next(a))


1
4
9
16
25
36
49
```

Generator Comprehension

- Or use generators defined in itertools

```
import itertools
squares2 = (n**2 for n in itertools.count(1,1))
```

```
>>> while True:
        print(next(squares2))


1
4
9
16
25
36
49
64
81
```

Generator Comprehension

- **WHY?**
  - Assume you want to process a huge set of data
  - You need to create intermediate results
  - If you use lists, they eat up memory
  - If you use generators, they don't

And now for something completely different

Copying Data Structures

- Copying and assignment are two different things

Copying Data Structures

- Copying and assignment are two different things

```
a = set(1, 2, "one")
```

  - We have an object a
  - We assign a to b
  - But the two objects are still linked:

Copying Data Structures

```
a = set([1, 2, "one"])
print(a)
b = a
print(b)
```

- Copying and assignment are two differen

```
# Now we change set a
a.remove("one")
# Which also changes set b
print(b)
```

```
>>> a = {1,2,"one"}
>>> a
{1, 2, 'one'}
>>> b = a
>>> a.remove("one")
>>> a
{1, 2}
>>> b
{1, 2}
```

Copying Data Structures

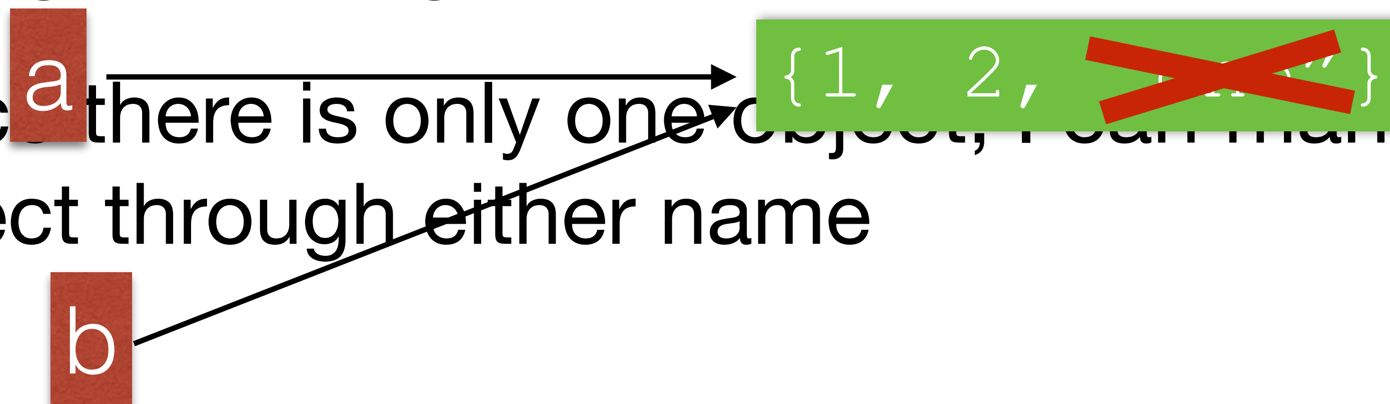
- Copying and assignment are two different things
  - Here is what happens
    - In Python, names point to objects
    - Assigning adds a name to the same object

a → {1, 2, "one"}

a → {1, 2, "one"}
b → {1, 2, "one"}

Copying Data Structures

- Copying and assignment are two different things
  - Since there is only one object, I can manipulate the object through either name



Copying Data Structures

- Copying and assignment are two different things
  - If I want to copy, I need to do so explicitly

```
lista = [1, 2, "three", [4,5]]
listb = [x for x in lista]
lista[2] = 3
print(lista)
print(listb)
```

```
>>> lista = [1, 2, "three", [4,5]]
>>> listb = [x for x in lista]
>>> lista[2] = 3
>>> lista
[1, 2, 3, [4, 5]]
>>> listb
[1, 2, 'three', [4, 5]]
```

- Now changes to one do not change the other!

Copying Data Structures

```
listb = lista[0:4]
```

- Copying and assignment are two different things
  - One can use slices to copy lists
  -

Copying Data Structures

- Copying becomes difficult if we have compound objects
    - E.g.: A list which contains lists, sets, …
- Shallow copy:
    - Resulting copies have shared elements

Copying Data Structures

- Example: A matrix as a list of rows
  - Create zero row by multiplying list with an integer

```
matrix = 3*[ 4*[0]]
```

- One might think it creates a structure like

```
[ [0, 0, 0, 0],
  [0, 0, 0, 0],
  [0, 0, 0, 0]]
```

  - which is not entirely false

Copying Data Structures

- We can get the elements as we should

```
matrix = 3*[ 4*[0]]
print(matrix[3][2])
```

- And we can set elements

```
matrix = 3*[ 4*[0]]
matrix[3][2] = 5
```

- But now we see that we got three times the same row

Copying Data Structures

```
matrix = 3*[ 4*[0] ]
print(matrix)
matrix[2][3] = 5
print(matrix)
```

```
RESTART: /Users/tjschwarzsj/Google Drive/AATeaching/Python/Programs/copying.py
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
[[0, 0, 0, 5], [0, 0, 0, 5], [0, 0, 0, 5]]
```

Copying Data Structures

- How can we do this:

```
matrix = [ [0 for _ in range(4)] for i in range(3)]
```

  - Need to construct the zero rows independently
    - Use e.g. list comprehension

Copying Data Structures

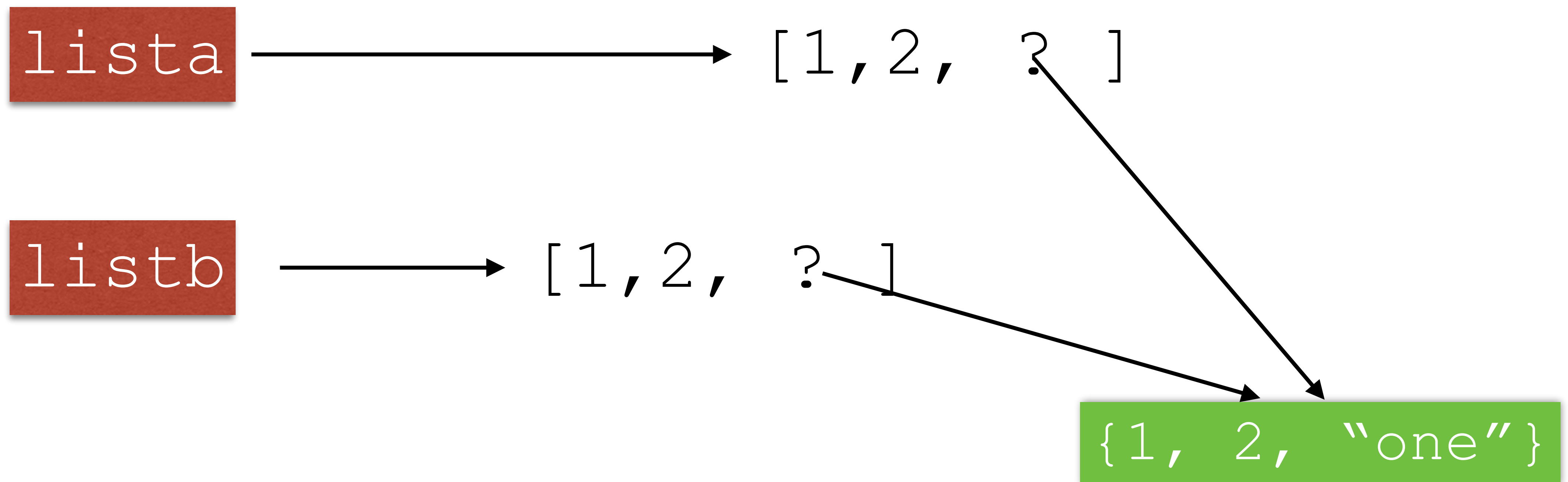
- Shallow copy: Assume we have

```
lista = [1, 2, [3,4,5]]
```

- We create a shallow copy by

```
lista = [1, 2, [3, 4, 5]]
listb = lista[:]
```

- But here is what is happening




```
lista = [1, 2, [3, 4, 5]]
listb = lista[:]
```

The two lists still share a componen
component in one list and change it

Copying Data Structures

lista → [1,2, ?]

- We have two copies of the list, but the third element are two d[illegible] names for the same object

listb → [1,2, ?]

{1, 2, "one"}

```
lista = [1, 2, [3, 4, 5]]
listb = lista[:]
```

Copying Data Structures

- In consequence, I can alter the same element in the list which is element number 2

```
lista = [1, 2, [3, 4, 5]]
listb = lista[:]
lista[2][0] = 6
print(lista)
print(listb)
```

- prints out

```
[1, 2, [6, 4, 5]]
[1, 2, [6, 4, 5]]
```

Copying Data Structures

- I need to use a deep copy
  - Easiest:
    - Use the module `copy`
      - Use `copy.deepcopy(object)` for deep copying
      - Use `copy.copy(object)` for shallow copying

Copying Data Structures

- This is a famous Python gotcha
    - Behavior is not intuitive.