PHYS 3210
9-23-19
Amanda Ash
P. 1

Exam 01

(1) General Python

(a) Similarities + differences between tuple, list, + Numpy array.

The tuple, list, and Numpy array can all store data, or other objects. For example, each of these objects could store a sequence of integers or floats. List's and Numpy array's can be changed and manipulated by their methods, or functions associated with the list and Numpy array objects, but tuples cannot be manipulated by ~~it~~ their methods. This is referred to as mutable and immutable objects respectively. Further, lists can contain different object types, but Numpy arrays must contain numbers of the same type (i.e. integers or floats).

(b) Difference b/w for + while loop. Which is best to avoid being stuck in infinite loop.

A for loop will perform an operation over every element, or iteration in a given range. For example, you could use a for loop to multiply every number in an array by 2, or you could perform a calculation 100 times using range(100). A while loop will perform an operation until a condition is met. As an example, you could use a while loop to add 1 to a number until the condition is met that the total sum is equal to 10. Because a condition must be met in order to exit a while loop, ~~t~~ while loops are more prone to becoming stuck in an infinite loop than for loops are. If the condition to break a while loop is never met, it will continue the operation forever, or until the program is terminated manually.

A. Ash
P. 2

Exam 01

(c) What is meant by everything in Python is an object? How does this make coding easier/harder

An object ~~is~~ characterizes everything in python by what information it can store + what operations it can do. For example, an array has information about its shape and can use functions such as reshape. Having objects can make computation faster as Python has built-in functions that take objects of the same type ~~and~~ as inputs. However, for these functions, you need to make sure the object types are all the same otherwise you can get ~~an~~ a type error. This means you may need to spend extra time/lines converting objects to different object types.

(d) Find an object and describe attributes and methods that could be assigned to it.

There is a bookshelf with 3 shelves, some trinkets on the shelves, a basket with scrap paper on the bottom shelf, and you guessed it, books on the shelves.

Attributes

(i) bookshelf.dimensions - ~~returns~~ information about length, height, and width of the bookcase = (~0.5m, 0.75m, 0.2m)

(ii) bookshelf.number_books - information about the number of books on the bookcase = 23

(iii) bookshelf.color - the color of the bookcase = (grey)

Methods

(i) bookshelf.append(object) - add something to the bookcase

(ii) bookshelf.set_shelfheight (height1, height2, height3) - move the shelves around in the bookcase

(iii) bookshelf.topple() - remove all the objects from the bookcase (rather violently)

Exam 01

(2) When integrating x^2 from $a=0$ to $b=10^4$, bin width $(h) \ll 1$, is it best to go from a to b or b to a?

When summing numbers, it is best to sum from small numbers to large numbers to avoid round off errors. Therefore, in this case it is best to use the limit definition of an integral, from a to b because $f(a) < f(b)$.

The reason it is best to perform a summation from ~~a~~ small numbers to large numbers is because the computer can only store so many significant figures. ~~As~~ Specifically, because Python uses double precision, it can store 16 decimal places. When Python stores a number, it does so in scientific notation, so every number has an associated exponent. When summing numbers, Python converts the exponent of the smaller number so that it is equal to the larger number. This is where the round-off error enters. To illustrate, we can look at an example where we want to sum:

$$1.0001 \times 10^{-15} + 10^4$$

$$= \underbrace{0.0000000000000000}\,00010001 \times 10^5 + 1.00000000000000000 \times 10^5$$

$$= 1.00000000000000000 \times 10^5$$

Because Python can only store so many significant figures if the smaller number is much smaller than the larger one, its contribution will be midigated.

We can check to see if this is the case by integrating this example both ways, from $a \to b$ and from $b \to a$. Doing this, we find that the sum from $a \to b$ is indeed more accurate to the analytic value.

A. Ash

Exam 01 P. 4

(3) is a Monte Carlo method appropriate? Calculate Velocity as a function of time considering a mass loss rate and effects due to air resistance. Then calculate the mass of fuel required to get the rocket into orbit.

A Monte Carlo method is not necessary for this problem. Rather, since air resistance and the mass loss rate are parameters which affect acceleration, we can use the finite difference method to calculate ~~veloc~~ acceleration in time steps, and use kinematic equations to update the velocity values. This gives us an approximation for velocity as a function of time as we can say:

Note: The finite difference method is useful to approximate ~~calculate~~ derivatives

$$\bar{a} = \frac{\Delta \bar{V}}{\Delta t} = \frac{\bar{F}}{m}$$, we consider air resistance in $\bar{F}_{net}$ and mass loss rate in m.

$$\therefore \bar{a} = \frac{(\bar{V}_i - \bar{V}_0)}{\Delta t} = \frac{\bar{F}}{m}$$

$$\therefore \bar{V}_i = \frac{\bar{F}}{m} \Delta t + \bar{V}_0$$, Velocity at step i is equal to the acceleration multiplied by a time step Δt plus the velocity from the previous step

The acceleration value will update as velocity is updated, then generating a new velocity.

Then to calculate the mass needed to get the rocket into orbit, you would set the final velocity equal to the velocity required to get the rocket to the radius of a geostationary orbit & calculate how many timesteps — or how much time was required to reach this velocity. Then we can use the mass loss rate from the fuel to determine how much fuel was required.

A. Ash
P.5

(4b)

V1

- all outcomes equally likely
- 7 possible outcomes
- $P_1 = P_2 = P_3 = P_4 + P_5 = P_6 = P_7 = \frac{1}{7}$

V2

- 4 addative outcomes equally likely
- 3 subtractive outcomes equally likely
- $P_{sub.} = \frac{1}{2} P_{add}$

Assume

(1) basket is initially empty
(2) There cannot be more than 10 cherries in the basket
(3) Assume one player

$1 = 4P_+ + 3P_-$ & $P_- = 0.5P_+$

$\therefore 1 = 4P_+ + \frac{3}{2}P_+$

$\therefore P_+ = \frac{2}{11} = P_1 = P_2 = P_3 = P_4$

$P_- = \frac{1}{11} = P_5 = P_6 = P_7$

```
turn_outcome = [ ]
  for turn in range(game) → spin the wheel over a range
    outcome = random number between 0 + 1
    if outcome ≤ P1 :
      ~~to turn~~ spin = +1  (add cherry)
      append turn_outcome (+1)
    if outcome ≥ P1 , ≤ P2 :
      spin = +2   (add 2 cherry)
      append turn outcome +2
    if outcome ≥ P2 , ≤ P3 :
      spin = +3  (add 3 cherry)
      append turn outcome (+3)
    if outcome ≥ P3 , ≤ P4 :
      spin = +4
      append turn outcome +4
    if outcome ≥ P4 , ≤ P5 :
      spin = -2 if sum(turn_outcome) ≥ 2
      spin = -1 if sum(turn_outcome) = 1 ← 1 cherry in basket
      spin = 0 if sum (turn_outcome) = 0 ← nothing in basket
      turn_outcome append(spin)
```

Exam 01

if outcome $\geq P_5, \leq P_6$:

spin = -2 if # cherries in basket ≥ 2

spin = -1 if # cherries in basket = 1

spin = 0 if # cherries in basket = 0

turn-outcome append (spin)

if outcome $> P_6$:

spin = - sum(turn-outcomes) if # cherries > 0

else

spin = 0 (no cherries in basket)

if sum(turn-outcome) ≥ 10

break → (the game is over)

return turn (# of turns to win)

<u>Note</u> :

- $P_1, P_2, P_3, P_4, P_5, P_6, P_7$ can be altered depending on game version
- add additional conditions to addative outcomes so the number of cherries in basket cannot exceed 10
- iterate over this scheme a large number of times to simulate many games

<u>Results</u> :

Simulating 10000 games of each version, version 1 where each outcome is equally likely requires ~ 2x as many turns to win than V2 where subtractive outcomes are half as likely as addative outcomes. Summizing the results :

	V1	V2
mean	15.1407	7.7418
median	11.0	6.0
std	12.741	5.481

Exam 01

M. Ash

P. 7

(4b cont) Assuming most children have a shorter attention span/ ~~tenacity~~ tenacity to become bored, I think the Version where all outcomes are equally probable is more suited for young children. This is because the game takes a ~~fewer~~ smaller number of turns to complete on average.