Python & R samenvatting

Iteratief = in de functie wordt weer verwezen naar zichzelf, basis waarde nodig anders gebeurt er niks. Efficiënt bij Fibonacci met O(n)

Recursief = met while of for loops. Tijd complexiteit O(2^n) voor Fibonacci. Voor tijd complexiteit kijk je dan naar lengte van binnenste loop keer de lengte van de buitenste loop. Inefficiënt omdat elk pad splitst en alles moet checken dan

Memoization = tussentijds opslaan van bepaalde waarden via dictionary. Er zit dan dictionary in het variabele gedeelte van de functie,

OO, object oriented approach in plaats van functie:

* Mogelijk dingen aan te passen zonder dat gebruik het ziet
* Gaat via bepaalde classes zoals in bijv. een game, in de class zitten dan functies.
  + Class heeft bepaalde properties, zoals lengte of kleur
  + Object is een exemplaar van die class, self refereert naar dat bepaalde object
  + Attribute is een waarde voor een specifiek object
  + Je hebt bijvoorbeeld de class Player, dit is de class en daaronder volgt dan def \_init\_(self, name) waarbij \_\_init\_\_ de **constructor** is en name is dan de **object,** de **attribute** zou dan bijvoorbeeld de naam Max zijn.
* Overloading: gebruik binnen class fraction dan def \_\_str\_\_ (self) en return eronder str(self.numerator)+’/’+str(self.denominator)
* Class variables zit gewoon in de class, instance variable zit in het functie gedeelte
* Methods = functies in een class
  + Instance method: bound to an object, can modify class and instance variables
  + Class method: can only access class variables, modify class state
  + Static method: can’t modify class or state, can’t modify class or isntance variables
  + Class en instance method zijn python **decorators**, zijn handig aangezien je dan functie of class kan aanpassen
* Child class, inherits alle eigenschappen van een parent of grandparent
* Super binnen de def van een child class refereert naar parent
* \_protected , \_\_ is private variables

Lecture 6

Knapsack problem, wil zo hoog mogelijk waarde met zo laag mogelijke weight binnen bepaalde capaciteit

1. Maakt een tabel met n + 1 rijen en w+ 1 kolommen, waarbij n = aantal items en w = capaciteit van de rugzak
2. Brute force checkt totale gewicht, waarde en of het boven capaciteit komt voor elke combinatie
3. Complexity O(n\*2^n)

Recursion approach

1. Voor elk item geldt, of het zit wel in de optimale combinatie of niet, check dit voor item n
2. Dan geldt in t eerste geval het optimale geval heeft waarde W, en in het tweede geval zal het dan waarde W – n hebben
3. Als weight n > W dan geldt sowieso dat n er niet in zit
4. Slaat elke waarde op in een tabel, allerlaatste m[n][w] is dan de hoogste
5. Complexiteit O(2^n)

Dynamical programming O(n x W) maarkost wel veel geheugen   
Greedy method vult naar de wat de beste value/weight ratio heeft wat doorgaat tot het de zak vult, heeft O(nlog(n)), geen gegarandeerde optimale oplossing maar wel simpel en snel

Branch and bround, soort brute force en checkt elke mogelijkheid maar kapt met een ‘tak’ wanneer boven max gewicht gaat

Github is shared software

Lecture 7, R

Data types:

* Numeric, gewoon een float
* Character, gewoon een string
* Logical, TRUE and FALSE

In plaats van = doe je <-, assignment operator

Functie c() maakt het een vector, binnen een vector moet alles van dezelfde class zijn

Bijv x <- c(1,3,4) dan class(x) = numeric, als c(x,y,z) met allemaal verschillende soorten vectors is de class logical < numeric < character als een van die dingen erin zit.

Matrix <- rbind combineert alle vectoren in een matrix met vectoren als rijen

Als xyz matrix daarvan is geeft xyz[, “z”] dan alleen kolom z, als daarna , drop = FALSE komt blijft met matrix met alleen waarde z erin

In een data frame kunnen verschillende classes, stel data frame ding heet d dan kan je element erin aanroepen met d$auto, handig voor kolommen met verschillende typen data

Roept element van lijst aan met dubbele haken

Lijst is handig om verschillende groepen objecten als data frames, matrix of variabelen in te stoppen.

Lecture 8

Rows are observations/cases and columns variables and elements are values, together makes tidy data

Ding voor het %>% is de buitenste, dus f(g(x)) becomes x %>% g() %>% f()

* Makkelijker te lezen en minder data