**1/12/18**

**List Generation (Construct List in nice way)**

* **method 1**
  + sqs = []
  + for i in range(10): sqs += [i\*i]
* **method 2**
  + sqs2 = [i\*i for i in range(10)]
  + no assignments to initialize or update list.
  + called **List Comprehension**

python’s polymorphism lets you use a variable as an int or string or whatever.

**Lambda Expressions**

* **method 1:** 
  + def cube(n): return n\*n\*n
* **method 2: (create method without a name and apply it directly0**
  + fourth = lambda n: n\*n\*n
* can embed lambda inside list comprehension
  + define a bunch of functions at once.

**Closure**

* a function that encapsulates the value(s) of some variable(s) that existed when the closure was created
* lambda is commonly used as a closure function

**Wicked Problem (Global challenge problems like ending homelessness)**

1. There is no definitive formulation of a wicked problem
2. Wicked problems have no stopping rule
3. Solutions to wicked problems are not true-or-false, but good-or-bad
4. There is no immediate and no ultimate test of a solution to a wicked problem
5. Every solution to a wicked problem is a “one-shot operation”; because there is no opportunity to learn by trial-and-error, every attempt counts significantly
6. Wicked problems do not have an enumerable (or an exhaustively describable) set of potential solutions, nor is there a well-described set of permissible operations that may be incorporated into the plan
7. Every wicked problem is essentially unique
8. Every wicked problem can be considered to be a symptom of another problem
9. The existence of a discrepancy representing a wicked problem can be explained in numerous ways. The choice of explanation determines the nature of the problem's resolution
10. The planner has no right to be wrong

**Why are problem formulations difficult**

Requires several kinds of knowledge.

* domain knowledge: about the problem
* about the problem-solving process
* about computer programming

Some problems are “ill-structured”.

* wicked problems, for example

**03-Basic-Search-Algorithms**

Iterative Deepening DFS

* We can combine the benefits of DFS and BreadthFS to get optimal paths without huge memory requirements.
* Instead of regular BreadthFS, we do a sequence of DFS executions, but with a depth limit for each execution. We make the depth limit increase by 1 in each execution, starting from 0.
* can save memory but not time efficient (repeated search of upper levels)

**Search Algorithms**

* Alternative objectives:
  + Reach any goal state
  + Find a short or shortest path to a goal state
* Alternative properties of the state space and moves:
  + Tree structured vs graph structured, cyclic/acyclic
  + Weighted/unweighted edges
* Alternative programming paradigms:
  + Recursive
  + Iterative
  + Iterative deepening
  + Genetic algorithms

**Weighted Edges**

* Cost of path: sum of the weights of its edges
* minimum-cost path: lowest weight cost path.

**Uniform-Cost Search**

* dijkstra's algorithm

**Heuristic**

* rule of thumb for operating in unknown, uncertain, or complex environments, or problem solving contexts.s