Contents

Recal

Iteration

Python handles iteration is a multitude of ways all of which basically accomplish similar tasks while utilizing different logic. For the purposes of this section we will utilize *for* loops. A for loop is a type of logic similar to the theory of ∀, in that it utilizes a collection for iteration. In a *for* loop there exist two basic ways to iterate, over a **custom range** or over a **list**.

```
for i in range(0,100):
  print(i)
```

In the above code there we use the **range** function to create a custom **list** (set) from zero to one hundred counting by one.

```
x = [0,1,2,3,4]
for i in x:
  print(i)
```

In the above code we have used a python **list** that we previously constructed. In both the code blocks the python code will act the same. Utilizing the *for* keyword we describe an iterator **i** that will represent each index of our preferred set. This type of iteration is useful when we want to traverse an entire list, say to find the sum?

Functions

```
def sum(l):
    ret = 0
    for i in l:
        ret = ret + 1
```

The above code is equivalent to the pseudo code found above In python we use the *def* keyword to denote a function, then we name it and place parenthesis containing any *parameters*. *Parameters* are the arguments that one can feed to a function for it to work properly. We can see that after the parameter list the colon symbol is used to denote the logics location beneath our declaration. The logic is of course a simple for loop with a temporary variable to contain the resulting values. *hint: There is a much shorter ways to form this function, extra credit for finding it, and bringing it to me.* For the more complex logic of course a more complex function is needed.

```python
import math

def euclideanDist(p0, p1):
    (x0, y0) = p0
    (x1, y1) = p1
    a = abs(x1 - x0)
    b = abs(y1 - y0)
    c = math.sqrt(a**2 + b**2)
    return c
```

While this function is in the scope of the class it is rather complex for the beginning of the semester and as such I will be bring it back up later. Line by line breakdown:

- **import math**: This imports the math library from the python standard library[1].
- **def euclideanDist(p0, p1):**: this declares the function **euclideanDist** and shows that it takes two parameters **p0** & **p1**, finally the colon symbol **:** directs us to the logic below.
- **(x0, y0) = p0**: This implies to python that **p0** is a tuple of the type **(int, int)** and extracts those values into **x0** & **y0** respectively.
- **a = abs(x1 - x0)**: This utilizes the absolute value function **abs()** in python, denoting that we want this result to be positive[2].
- **c = math.sqrt(a2 + b2)**: This is utilizing that library we imported earlier to invoke the square root function **sqrt**. The ** symbols here represent the *exponent* operation.

Software dev life cycle *SDLC* (brief)

Plan

This is where we plan infrastructure like compute, storage, …, etc(hardware requirnments)

Design

plan the software.

Implement

Coding phase (build the thing).

Test

bit obvious, the software is tested.

Deploy

Deployment is heavily dependant on the platform and type of software but usually composes of some build pipeline, accompanied by a **compiled** artifact being hosted on a server.

Maintain

- Fix the bugs
- Answer user complaints
- act on improvement plans

Code conventions

Code conventions are etiquette of a code base, usually defined by the owner or maintainer. In modern times the languages we use will specify most of them, however some code bases will still

[1]later in the class we will come back to the python standard library
[2]recall that distance is always positive

use their own even with those languages that specify them. Conventions more than anything else are a way to make code easier to read over long periods of time.

Comments

Comments are a way to insert inert English into your code for the exclusive purpose of explaining how the logic works. In Python we use the '#' symbol to denote single line comments.

```
x = 0 # variable 'x' is equivalent to zero
def plus(x,y):
    return x + y # returns the simple addition of two variables
```

Naming

The naming of variables and functions can be regulated to preserve code regularity.

```
nameOfVariable = 0
def name_of_function(x):
  print("snake_case")
```

We observe in the code above that variables names use a style known as *camalCase*. The style camalCase denotes a style of zero space naming that capitalizes the first letter of every word except the first one. For example nameOfVariable capitalizes both the **O** & **V** characters but not the **n** character. Usually if there exist naming conventions there will be extensive documentation on such conventions (papers, rules, people that grumble when you forget).

external Documentation

Externals docs will almost always consist of papers that are completely separate from the code. These will rarely (unfortunately) be academic research. Usually these papers are architect notes or designs like diagrams, etc.

Nomenclature

Σ l - is the *sumation* of the set l, this means that it takes each element of all and preforms the addition operation on them sequentially
$s.t$ or : - read "such that"
$\forall$ x $\in \mathbb{Z}$ - read "for all x in $\mathbb{Z}$", denotes iteration over all indices of $\mathbb{Z}$
A $\longleftarrow$ B - denoting a binding where A will represent the value of B
A $\longrightarrow$ B - read A **implies** B, denoting the that the statement A being true implies that B is also true
[ ] - denoting a set
$\mathbb{Z}$ - set of all integers (real numbers or their negatives)
$\mathbb{R}$ - set of all real numbers (rational numbers/irrational numbers, including fractions and their negatives)
i $\in$ I - denotes that i is an element of the set I