More on *docstrings*

Describe what a function does, be specific

- Mention all parameters by name
- Must not include how the function works
- No mention of local variables, implementation details (algorithms, helper methods, etc.)

Docstrings-purposes

- Defines an interface => callers know how to use it
- Helps you implement the body and meet the specs
- Helps with debugging and code maintenance

Implementation

```python
def length_is_multiple(string, num):
    """
    Return whether the length of the given string is
    multiple of num

    @param str string: a string
    @param int num: a whole number
    @rtype:bool

    >>> length_is_multiple("two",3)
    True
    >>> length_is_multiple("two",2)
    False
    """
    return len(string) % num == 0
```

Two types of docstrings

1. Epytext (we are using this) `@param str string: a string`
2. reStructuredText (default in Pycharm) `:param str string: a string`

<u>Change in Pycharm:</u>

Preferences -> Tools -> Python Integrated Tools -> Docstring format: Epytext

apply

Pre/post conditions

```
def square_root(number):
  """Calculate the square-root of <number>
  @type number: int
  @rtype: float

  @precondition: number >= 0
  @postcondition: abs(res * res - number) < 0.01

  <Usage examples ...>
  """
  assert number >= 0, "Uh-oh, invalid input"
  res = sqrt(number)
  assert abs(res * res - number) < 0.01
  return res
```

Design contract - summary

A binding agreement with the client

- Given a set of preconditions, a set of promised results will occur
- If not => no guarantees!

For a function, if the arguments satisfy the type contract and the preconditions, then the function:

- Will not crash
- produces the expected result

A story of a “type”

Type hinting in Python

Introduced in Python3.5

Python is a *dynamically* typed language

- Type of a variable determined at runtime
- E.g. a="hello" (a is **str**)
- def sum (a, b): sum (3, 4) → a, b : int but sum('3', 4) → a: str and b: int

Type hinting allows checking for types without running the code (*statically*)

- Using tools like *mypy* (`python3.6 -m pip install mypy`)

Type hinting

```python
def length_is_multiple(string, num):
    """
    Return whether the length of the
    given string is multiple of num

    @param str string: a string
    @param int num: a whole number
    @rtype:bool
```

```python
def length_is_multiple(string: str, num: int) -> bool:
    """
    Return whether the length of the
    given string is multiple of num

    @param str string: a string
    @param int num: a whole number
    @rtype:bool
```

Python3.6

We will be using this notation

Data abstraction, objects

Python: Everything is Object

```
>>> type(10)
<class 'int'>
>>> type('str')
<class 'str'>
>>> type(1.56)
<class 'float'>
```

```
>>> def inc(e): return e+1
...
>>> type(inc)
<class 'function'>
```

```
>>> A = 0
>>> type(A)
<class 'int'>
```

Classes and objects

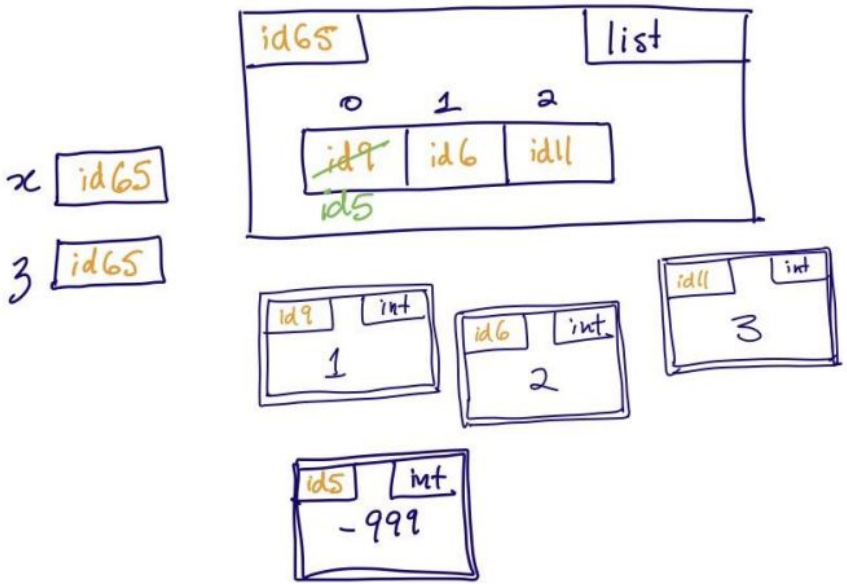
- What's a class?
  - Abstract data structure that models a real-world concept
  - Describes the attributes and "abilities" (methods) of that concept (called object)
  - Example: int, str, list, etc., or user-defined: Point, Rectangle, Cat, Desk, FileReader, ColourPrinter, etc.
- What's an object?
  - Instance of a class
  - Everything in Python is an object!

An object has 3 components

- id (a reference/alias to its address in memory)
- data type (defines what they can do)
- value

Memory Model

```
>>> x = [1, 2, 3]
>>> z = x
>>> z[0] = -999
```



Different data types

Immutable:

Once stored in memory, it cannot change!

e.g., integers, booleans, strings, etc.

Mutable:

A type that is not immutable

e.g., lists, dictionaries

Equality

Equality of values in memory: ==

Equality of addresses in memory: is

Examples:

A = 1000

A == 1000 True

A is 1000 False

Equality

Equality of values in memory: ==

Equality of addresses in memory: is

Examples:

A = 1000	A = 5
A == 1000 True	A == 5 True
A is 1000 False	A is 5 True (!!)

python caches integer values in the range (-5, 256)

Class level methods

```
class Student:

    course: str

    def __init__(self):
        self.course = ""

    def enrol(self, course_given: str) -> None:
        self.course = course_given
```

```
s = Student()
s.enrol("CSC148H1s")
```

The object “s” automatically passed as first argument of enrol().

Now we will design a class

It’s about the simplest geometric shape:

A point

Definition of Point

In two dimensions, a point is two numbers (coordinates) that are treated collectively as a single object. Points are often written in parentheses with a comma separating the coordinates. For example, (0, 0) represents the origin, and (x, y) represents the point x units to the right and y units up from the origin. Some of the typical operations that one associates with points might be calculating the distance of a point from the origin, or from another point, or finding a midpoint of two points, or asking if a point falls within a given rectangle or circle.

Designing a Point

Analyzing specification

In two dimensions, a point is two numbers (coordinates) that are treated collectively as a single object. Points are often written in parentheses with a comma separating the coordinates. For example, (0, 0) represents the origin, and (x, y) represents the point x units to the right and y units up from the origin. Some of the typical operations that one associates with points might be calculating the distance of a point from the origin, or from another point, or finding a midpoint of two points, or asking if a point falls within a given rectangle or circle.

Designing a Point

In two dimensions, a point is two numbers (coordinates) that are treated collectively as a single object. Points are often written in parentheses with a comma separating the coordinates. For example, (0, 0) represents the origin, and (x, y) represents the point x units to the right and y units up from the origin. Some of the typical operations that one associates with points might be calculating the distance of a point from the origin, or from another point, or finding a midpoint of two points, or asking if a point falls within a given rectangle or circle.

(3,4) (-2,3) (5,0)

Designing a Point

In two dimensions, a point is two numbers (coordinates) that are treated collectively as a single object. Points are often written in parentheses with a comma separating the coordinates. For example, (0, 0) represents the origin, and (x, y) represents the point x units to the right and y units up from the origin. Some of the typical operations that one associates with points might be calculating the distance of a point from the origin, or from another point, or finding a midpoint of two points, or asking if a point falls within a given rectangle or circle.