COL1000: Introduction to Programming

REVIEW CLASS

Subodh Sharma | Lec 13 | Sept 02

Recognising Common Errors (RECAP)

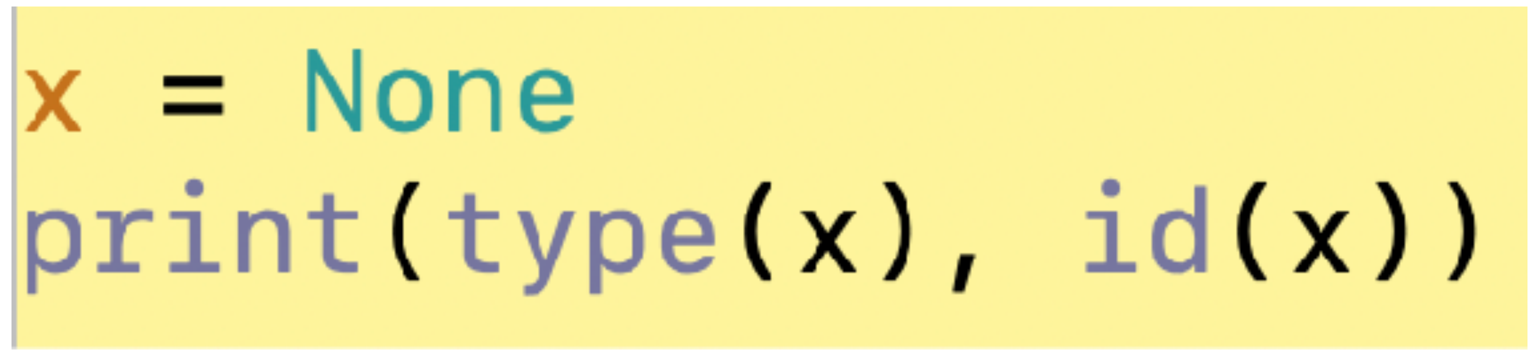
- Indentation Error: Not indented properly
- Name Error
- Value Error
- Type Error
- Attribute Error
- Index Error
- Key Error
- DivByZero Error

Objects: Mutability (RECAP)

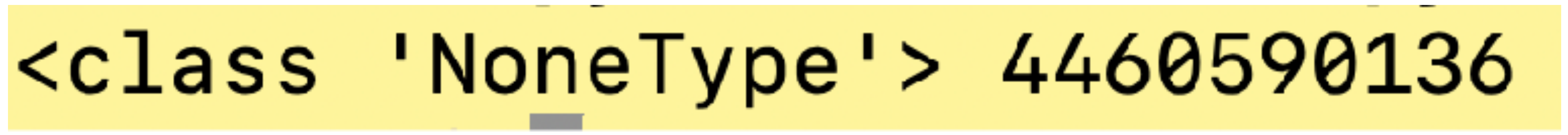
- **Mutable objects**: Objects whose contents can be changed without changing their identity (i.e id remains the same)
    - Examples: lists, dict, sets
- **Immutable objects:** Objects once created cannot be modified
    - Examples: int, float, bool, str, tuple

Objects & Types (RECAP)

- Recall each object has a: **type**, **value**, **Id**
  - Use `type()`, `id()` to obtain type and Id information regarding the object



```
x = None
print(type(x), id(x))
```



```
<class 'NoneType'> 4460590136
```

  - Named variables for objects: **act just as labels**
  - Is vs == : Is operator checks the equality over Ids of its operands (`x is y` ≡ `id(x) == id(y)`); == is check of equality over values of its operands
  - `x += v` and `x = x+v` **are subtly different**; former is in-place and doesn't create a new object while the latter does

References (RECAP)

- `a = b` (`b`’s reference is copied in to `a`)
- `a = b + 5` (fetch b’s value through its reference and evaluate the expression)
- `l1 = [‘5’, 10, a]` and then `l2 = l1`
    - `print(id(l1) == id(l2)) —> True`
    - But lists are mutable. `l1.append(“svs”); l1[3] = l1; l2 = l1 + l1`
        - Check now if the ids are the same?
- `x,y = v1, v1 (`assign references from left to right`)`
- `x,y = [5, “svs”]` (unpacking of the object to constituents and then binding them to the references
- `l1 = [‘5’, 10, a]; l2 = l1; del l1` —> l1 name is deleted but the object is alive through reference l2

Type & Conversion

- `int(float_data)`: truncates towards 0
- `round(x)` : rounds to the nearest integer (round half to nearest even)
- `math.floor(x)`: truncates towards ∞
- `Str(mostly_any_type_of_data)`: converts input data to string
- `tuple(lst):` Converts list to *immutable* tuple
- `set(lst)`: removes duplicates and gives an *unordered* collection

Containers: Lists, Tuples, Strings

Lists

Ordinal Containers

- Each list item has an ordinal position
- List functions
  - Append: `lst1 = [1,2,3]; ls1.append(“e”) # lst1 = [1, 2, 3, “e”]`
  - Insert: `lst1.insert(1, “svs”) # lst1 = [1, “svs”, 2, 3, “e”]`
  - Remove: `lst1.remove(“e”) # lst1 = [1, “svs”, 2, 3]`
  - Reverse: `lst1.reverse() # lst1 = [3, 2, “svs”, 1] – done in-place`
  - Sort: `lst1.sort() # lst1 = TypeError, < is unsupported for str and int`
    - If lst1 `= [2,1,5,3]` then `lst1.sort() = [1,2,3,5]`
  - Sort (reverse): `lst1.sort(reverse = true) # lst1 = [5,3,2,1]`

Lists

Shallow Copy vs Deep Copy

- **Shallow copy** — creates a new object, but reuse the references for internal entities
  - `A = [1,2, [3,4]]; B = A.copy(); id(A) != id(B); id(A[2]) == id(B[2])`
- **Deep copy** — create new object and recursively all entities internally — nothing is shared!
  - `A = [1,2, [3,4]]; import copy; B = copy.deepcopy(A); id(A) != id(B); id(A[2]) != id(B[2])`

Lists

Slicing

- `lst1 = [x for x in range(6)] # lst1=[0,1,2,3,4,5]`
- `lst1[2:5] # [2,3,4] – up to but excluding the “to” param`
- `lst1[:3] # [0,1,2]`
- `lst1[::2] # skips every 2nd element; [0,2,4]`
- `lst1[::-1] # reverses the list`
- `lst1[5:2:-1] # ?`

Lists

Combinations, Comprehensions, Built-in functions

- `lst1 = [[0]*3] # lst1=[[0,0,0]]`
- `A + B: concatenation of lists`
- `lst1=[x*x if x > 0 else -x*x for x in range(5)]`
- `min, max, len functions`

Tuples

- Ordered, immutable, heterogeneous container
  - Tuple id remains the same; but internal elements, if mutable, are allowed to change
- `head, *mid, tail = (1, 2, ‘svs’, 3, 4)`
  - `head = 1; tail = 4; mid = [2, ‘svs’, 3]`
- `_,y = (25, 46) # _ is used for ignoring during unpacking`
- Comprehensions work just the same as they do for lists
- min, max, reverse, sort functions work just same with a caveat:
  - `sorted(tup) # produces a list`
  - `tuple(sorted(tup)) # produces a sorted tuple`

Tuples

- Ordered, immutable, heterogeneous container
  - Tuple id remains the same; but internal elements, if mutable, are allowed to change
- `head, *mid, tail = (1, 2, ‘svs’, 3, 4)`
  - `head = 1; tail = 4; mid = [2, ‘svs’, 3]`
- `_,y = (25, 46) # _ is used for ignoring during unpacking`
- Comprehensions work mostly the same as they do for lists except they produce generator expression
  - Have to be explicitly typed into Tuples: Eg: `tuple(x*x for x in range(5))`
- min, max, reverse, sort functions work just same with a caveat:
  - `sorted(tup) # produces a list`
  - `tuple(sorted(tup)) # produces a sorted tuple`