CS1010S

Tutorial 5: Data Abstraction

Nicholas Russell Saerang (russellsaerang@u.nus.edu)

Table of contents

Recap

Slice operator

Tuple

Immutability

Slice operator [::]

Slicing

Slicing

Given an iterable object **seq** (str, tuple)

You can slice it using **seq[start:end:step]**

The start, end and step is similar to range

- **Inclusive** start
- **Exclusive** end
- Any integer step, use only **positive integer** in CS1010S

Slicing and range

```
>>> tup1 = (1, 2, 3, 4, 5)
>>> str1 = '12345'
>>> tup1[0:4:2] # Include at 0, exclude at 4
    (1, 3)
>>> str1[0:4:2] # Works on string
    '13'
>>> for i in range(0, 4, 2): # Same behavior as range
        print(tup1[i])
>>> 1
    3
```

Tuples

Immutable sequences

Tuple

A **immutable** sequence of elements

Is **reference** type

Element could be any type

Syntax: (..., ...,)

Always remember the COMMA (...,)

```
(1, 2,) # Looks weird but ok
(1, 2)  # Normal
(1,)    # Comma needed
```

Tuple

A **immutable** sequence of elements

Is **reference** type

Element could be any type

Syntax: (..., ...,)

Always remember the COMMA (...,)

```
(1, 2,) # Looks weird but ok
(1, 2)  # Normal
(1,)    # Comma needed
```

Primitive Type: (int, str, float, bool, none)

```
a = "same"
b = "same"
a == b # True
a is b # True

x = 257
y = 257
x is y # False (weird behavior)
```

Reference Type:

- Look alike (!⇒) Same Identity
- Same Identity ⇒ Look alike

```
tup1 = (1,2)
tup2 = (1,2)
tup1 == tup2 # True
tup1 is tup2 # False
```

Modifying Tuples

```
How to append to tuples?
(1, 2, 3, 4) -> (1, 2, 3, 4, 5)

>>> seq1 = (1, 2, 3, 4)
>>> seq2 = seq1 + (5, )

>>> seq2
    (1, 2, 3, 4, 5)
```

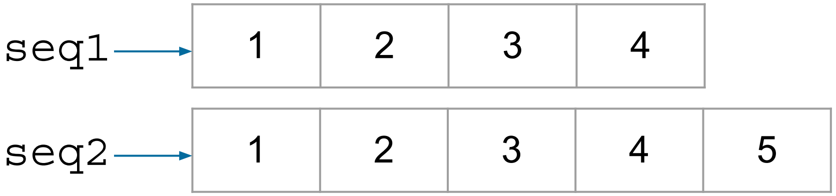
Modifying Tuples

How to append to tuples?

(1, 2, 3, 4) -> (1, 2, 3, 4, 5)

```
>>> seq1 = (1, 2, 3, 4)
>>> seq2 = seq1 + (5, )

>>> seq2
    (1, 2, 3, 4, 5)
```



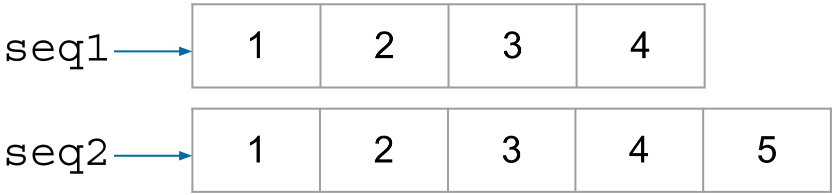
Modifying Tuples

How to append to tuples?

(1, 2, 3, 4) -> (1, 2, 3, 4, 5)

```
>>> seq1 = (1, 2, 3, 4)
>>> seq2 = seq1 + (5, )

>>> seq2
    (1, 2, 3, 4, 5)
```



seq1 is unchanged due to **immutability**

A **new tuple** seq2 is created

Iterating Sequences

```
seq = tuple(range(0, 4, 1)) # (0, 1, 2, 3)
```

Two ways to iterate through seq

- Iterate through elements

```
for i in seq:
    print(i)
```

- Iterate through indices

```
for i in range(0, len(seq), 1):
    print(seq[i])
```

Which do you prefer?

Abstract data types

Encapsulation

ADT Behaviour

Internal implementation of ADT is hidden from user

- Only need to know expected behaviour

ADT Behaviour

Internal implementation of ADT is hidden from user

- Only need to know expected behaviour

Getters and setters provide **interface** to data structures.

```
make_point(x,y) → (x, y)
get_x(point)    → x_coor_of_point
get_y(point)    → y_coor_of_point


get_x(make_point(2,9)) == 2
get_y(make_point(2,9)) == 9

# You can use it WITHOUT knowing HOW it is implemented!
```

ADT Behaviour

Getters and setters provide interface to data structures.

```
# Suppose this is the implementation
# (don’t use this for exams!!!)
make_point = λx,y: λs: x if s == 0 else y
get_x = λp: p(0)
get_y = λp: p(1)

# You might not even understand what is the code doing!
# But you as a user, just need to know how to use!
```

Tutorial 5

Data Abstraction

Question 1: Midpoint segment

(a) In this question, you will implement a representation of line segments in a 2D plane. Some sample executions are provided for you to test your implementations but you are **strongly encouraged** to create your own test cases.

i. A point can be represented as a pair of numbers: the x coordinate and the y coordinate.

Based on this representation of a point

- Implement a point constructor `make_point`.
- Implement a selector `x_point` which returns the x coordinate of a given point.
- Implement a selector `y_point` which returns the y coordinate of a given point.

Question 1: Midpoint segment

ii. A line segment has two endpoints. It can be represented by a pair of points: a starting point and an ending point.

Based on this representation of a line segment, implement a line segment constructor `make_segment`.

Implement a selector `start_segment` which returns the starting point of a given line segment.

Implement a selector `end_segment` which returns the ending point of a given line segment.

Question 1: Midpoint segment

iii. Finally, using the selectors and constructors which you have implemented, implement a function `midpoint_segment` that takes a line segment as argument and returns its midpoint. (The midpoint of a line segment is the point whose coordinates are the averages of the coordinates of the endpoints of the line segment.)

Sample Runs (continued from **??***):*

```
m = midpoint_segment(s)
print_point(m) #expected printout: ( 3.5 , 5.0 )
```

Question 1: Midpoint segment

```
Defined just now!

make_point(x, y) → point
x_point(point) → number
y_point(point) → number
```

Question 1: Midpoint segment

```
make_segment(point1, point2) → segment
start_segment(segment) → point
end_segment(segment) → point

def make_segment(point1, point2): # → segment
    return (point1, point2)

def start_segment(segment): # → point
    return segment[0]

def end_segment(segment): # → point
    return segment[1]
```

Question 1: Midpoint segment

```
make_segment(point1, point2) → segment
start_segment(segment) → point
end_segment(segment) → point

def midpoint_segment(s): # → point
    s1, s2 = start_segment(s), end_segment(s)
    x = (x_point(s1) + x_point(s2))/2
    y = (y_point(s1) + y_point(s2))/2
    return make_point(x, y)
```

Question 1: Rectangles

(b) Implement a representation for a rectangle in a 2D plane. In terms of your constructors and selectors, create functions that compute the perimeter and the area of a given rectangle.

Now implement an alternative representation for rectangles. Can you design your system with suitable abstraction barriers, so that the same perimeter and area functions will work using either representation?

Question 1: Rectangles

Rectangles can be defined in terms of their length/width, or their coordinates of their corners.

```
def make_rect1(l, w):
    return (l, w)

def make_rect2(tl, br):
    return (tl, br)
```

Question 1: Rectangles

How to know which abstraction being used? We can tag it!

```
def make_rect1(l, w):
    return (l, w, "side")

def make_rect2(tl, br):
    return (tl, br, "pt")

def get_tag(rect):
    return rect[-1]

def get_value(rect):
    return rect[0:2:1]
```

```
def area(rect):
    if get_tag(rect) == "side":
        return area1(get_value(rect))
    elif get_tag(rect) == "pt":
        return area2(get_value(rect))
```

Question 2: Product Data Type

(a) Your first task is to design a `Product` data type, which serves to model various perishable goods in the convenience store. The `Product` data type supports the following functions:

- `make_product(name, shelf_life)` takes the name of the product (**a string**), and number of days (**an integer**) that the product can remain on the shelf before expiring. It returns a data type representing a `product`.
- `get_name(product)` takes in a `product`, and returns its name.
- `get_shelf_life(product)` takes in a `product`, and returns its shelf life.

Decide on a data structure **using tuples** to represent a record, and implement the functions `make_product`, `get_name`, and `get_shelf_life`.

Question 2: Product Data Type

Decide on a data structure **using tuples** to represent a record, and implement the functions `make_product`, `get_name`, and `get_shelf_life`.

```python
def make_product(name, shelf_life):
    return (name, shelf_life)

def get_name(product):
    return product[0]

def get_shelf_life(product):
    return product[1]
```

Question 2: Inventory Data Type

(b) An `Inventory` is a collection of `Product` objects in the convenience store. The `Inventory` data type supports the following functions:

- `new_inventory()` returns a data type representing an `inventory` that does not contain any product.
- `add_product(inv, product)` takes in an `inventory` and a `product`, and returns a new inventory with a product added to it.
- `add_one_day(inv)` takes in an `inventory` and returns a new inventory with the same products having sat one more day on the shelf.
- `get_expired(inv)` takes in an `inventory` and returns a tuple of products that have sat on the shelf longer than their shelf life. Note that the products returned should be **equivalent to the products** added to the inventory. See sample execution for details.

Question 2: Inventory Data Type

Decide on a data structure **using tuples** to represent an inventory. Provide an implementation for the functions new_inventory, add_product, add_one_day and get_expired.

```
def make_inventory():
    return ()

def add_product(inv, product):
    return inv + ((product, 0), )
```

The plan is to represent an inventory as a tuple of (product, days_in) pairs, so every new product has 0 days in.

Question 2: Inventory Data Type

```
def add_one_day(inv):
    res = ()
    for p in inv:
        res += ((p[0], p[1] + 1), )
    return res

def get_expired(inv): # → (product1, product2, … )
    res = ()
    for p in inv:
        if get_shelf_life(p[0]) < p[1]:
            res += (p[0], )
    return res
```

Question 2: Inventory Data Type

```
You will learn lambda, map, and filter this week!
Very useful tools in Python!

def add_one_day(inv):
    return tuple(map(lambda p: (p[0], p[1] + 1), inv))

def get_expired(inv): # → (product1, product2, … )
    return tuple(map(lambda p: p[0],
                     filter(lambda p: get_shelf_life(p[0]) < p[1],
                            inv)))
```

Extra Question

- How do you get the minimum number of days until at least one product in the inventory expires?

- How do you get the tuple of all **distinct product names**?

The End