CS1010S

Tutorial 6: Working with Sequences and Lambda Functions

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

Table of contents

Lambda

Concise function definition

Lambda / Anonymous Function

Return value must be computable as a single expression

$$x \mapsto x^2$$

```
def square(x):
    return x**2 # single expression
```

Lambda / Anonymous Function

Return value must be computable as a single expression

$$x \mapsto x^2$$

```
def square(x):
    return x**2 # single expression

square = lambda x: x**2
```

Lambda / Anonymous Function

Return value must be computable as a single expression

$$x \mapsto x^2$$

```
def square(x):
    return x**2 # single expression

square = lambda x: x**2

# square(3) == (lambda x: x**2)(3) == 3**2 == 9
```

def vs lambda

You can take it as an alternative approach to define a function.

The def allows you to define a function in multiple lines

The lambda allows to you define a function in one line

Lambda expressions

When do we use lambda expressions?

Usually when we are only using the function once, for example feeding it into map/filter.

```
iterable = range(0, 11, 1)
tuple(map(lambda x: x**2, filter(lambda x: x%2==0, iterable)))
```

Advantage of lambda

```
iterable = range(0,11,1)
squared_evens = tuple(map(lambda x: x**2,
                          filter(lambda x: x%2==0, iterable)))
```

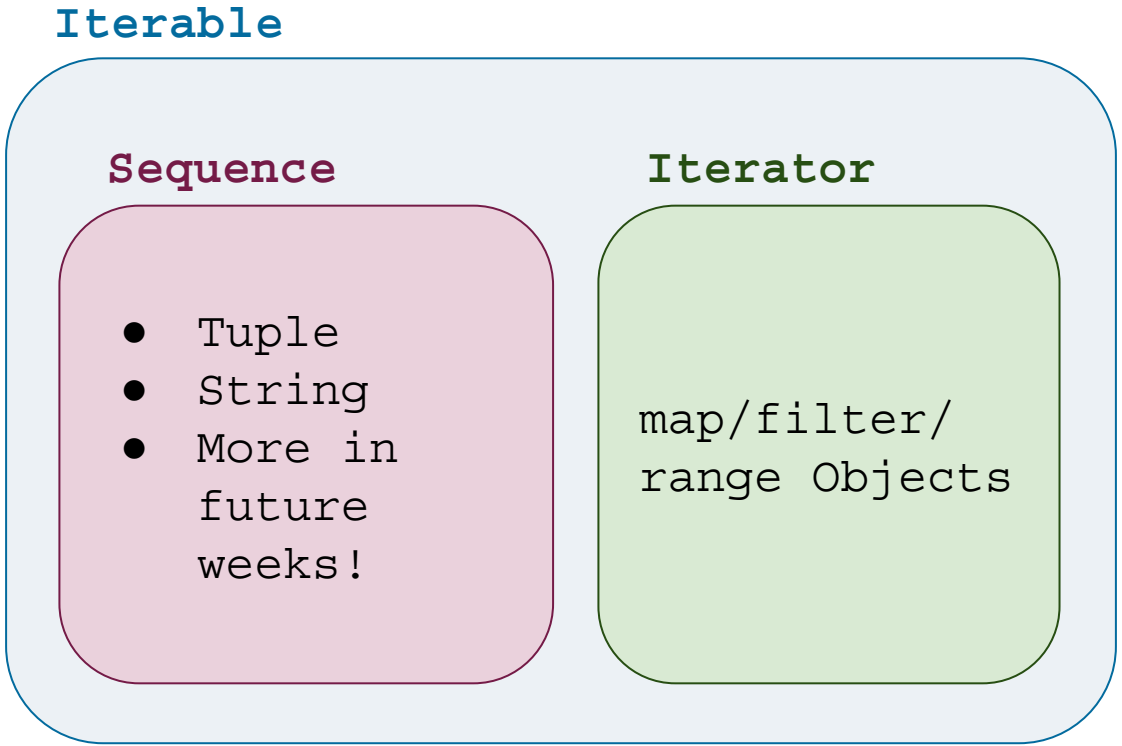
VS

```
iterable = range(0, 11, 1)
squared_evens = ()
for x in iterable:
    if x % 2 == 0:  # Filter: even x
        squared_evens += (x ** 2,)  # Map: square x
```

Map & Filter

Operations on tuple

Terminology



Map

Map a function on all values of an iterable

Syntax: map(function, iterable)

Example:

```
x = (1, 2, 3)
```

Add 1 to all values in x

```
function: lambda i: i + 1
iterable: x
```

Filter

Filtering by a condition

Syntax: filter(function, iterable)

Example:

```
x = (1, 2, 3)

Filter only odd values in x

function: lambda i: i % 2 != 0 # is_odd
iterable: x
```

Map Object and Filter Object

map/filter returns a map/filter object.

Always remember to explicitly convert the object back to the desired data type.

```
>>> map(lambda x:2*x, range(0,11,1))
    <map object at 0x000001A61AEE3250>
```


```
>>> tuple(map(lambda x:2*x, range(0,4,1)))
    (0, 2, 4, 6)
```

Map Object and Filter Object

```
only_once = map(lambda x:x, range(0,11,1))
>>> <map object at 0x000001A61AEE3250>

for elem in only_once: # first run
    print(elem) # it will print 0, 1, 2, ..., 10

for elem in only_once: # second run
    print(elem) # it will print nothing after the first run
```

Map Object and Filter Object

```
infinite_time = tuple(map(lambda x:x, range(0,4,1)))
>>> (0,1,2,3)

for elem in infinite_time: # first run
    print(elem) # it will print 0, 1, 2, 3

for elem in infinite_time: # second run
    print(elem) # it will print 0, 1, 2, 3

for elem in infinite_time: # third run
    print(elem) # it will print 0, 1, 2, 3
```

- **In short, always remember to explicitly typecast**
-
-

Tutorial 5

Working with Sequences

Question 1: Odd indices

Write a Python function called **odd_indices** that takes in a tuple as its only argument and returns a tuple containing all the elements with odd indices (i.e. every second element from the left) from the input tuple. For example:

```
>>> odd_indices (( 'a', 'x', 'b', 'y', 'c', 'x', 'd', 'p',
'q'))

('x', 'y', 'x', 'p')
```

Question 1: Odd indices

```
def odd_indices(tup):
    new_tup = ()
    for i in range(1,len(tup),2):
        new_tup += (tup[i],)
    return new_tup
```

Question 1: Odd indices

```
def odd_indices1(tup): # recursion
    if len(tup) < 2:
        return ()
    return (tup[1],) + odd_indices1(tup[2:len(tup):1])

def odd_indices2(tup): # slicing
    return tup[1:len(tup):2]
```

Question 1: Odd indices

```
def odd_indices3(tup): # enumerate-filter-map
    return tuple(map(lambda x: x[1],
                     filter(lambda x: x[0]%2 == 1,
                            tuple(enumerate(tup)))))
```

enumerate is similar to map and filter but transforms an element x to a pair (index, x)

e.g.

```
>>> tup = (2, 10, -7)
>>> print(tuple(enumerate(tup))
    ((0, 2), (1, 10), (2, -7))
```

Question 2: Even/odd sum

4. Write a function called **even_odd_sums** that takes in a tuple of numbers as its only argument and returns a tuple of two elements: the first is the sum of all even-indexed numbers in the input tuple, while the second element is the sum of all odd-indexed elements in the input tuple.
   Example execution:

```
>>> even_odd_sums((1, 3, 2, 4, 5))
(8, 7)
>>> even_odd_sums((1,))
(1, 0)
>>> even_odd_sums(())
(0, 0)
```

Question 2: Even/odd sum

```
def even_indices(tup):
    return odd_indices((0,) + tup)

def odd_even_sums(tup):
    return (sum(even_indices(tup)),
            sum(odd_indices(tup)))
```

Question 3: Manipulating Sequences with HOF

Suppose x is bound to the tuple (1, 2, 3, 4, 5, 6, 7).

Using map, filter, and lambda expressions.

Question 3: Manipulating Sequences with HOF

```
x = (1, 2, 3, 4, 5, 6, 7)

# (a) (1, 4, 9, 16, 25, 36, 49)
```

Question 3: Manipulating Sequences with HOF

```
x = (1, 2, 3, 4, 5, 6, 7)

# (a) (1, 4, 9, 16, 25, 36, 49)

a = tuple(map(lambda i: i**2, x))
```

Question 3: Manipulating Sequences with HOF

```
x = (1, 2, 3, 4, 5, 6, 7)

# (a) (1, 4, 9, 16, 25, 36, 49)

a = tuple(map(lambda i: i**2, x))

# (b) (1, 3, 5, 7)
```

Question 3: Manipulating Sequences with HOF

```
x = (1, 2, 3, 4, 5, 6, 7)

# (a) (1, 4, 9, 16, 25, 36, 49)

a = tuple(map(lambda i: i**2, x))

# (b) (1, 3, 5, 7)

b = tuple(filter(lambda i: i % 2 == 1, x))
```

Question 3: Manipulating Sequences with HOF

```
x = (1, 2, 3, 4, 5, 6, 7)

# (a) (1, 4, 9, 16, 25, 36, 49)

a = tuple(map(lambda i: i**2, x))

# (b) (1, 3, 5, 7)

b = tuple(filter(lambda i: i % 2 == 1, x))

# (c) ((1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (7, 7))
```

Question 3: Manipulating Sequences with HOF

```
x = (1, 2, 3, 4, 5, 6, 7)

# (a) (1, 4, 9, 16, 25, 36, 49)

a = tuple(map(lambda i: i**2, x))

# (b) (1, 3, 5, 7)

b = tuple(filter(lambda i: i % 2 == 1, x))

# (c) ((1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (7, 7))

c = tuple(map(lambda i: (i, i), x))
```

The End