Theory of Computation and Programming Languages

Higher-order functions

CS51 – Spring 2026

Higher order functions

- Have you ever called a function but forgot the parentheses? For example,

```
def add_one(x):
    return x+1
>> add_one
<function add_one at 0x108e962f0>
```

- Notice that it does not give an error.

```
>>> type(add_one)
<class 'function'>
```

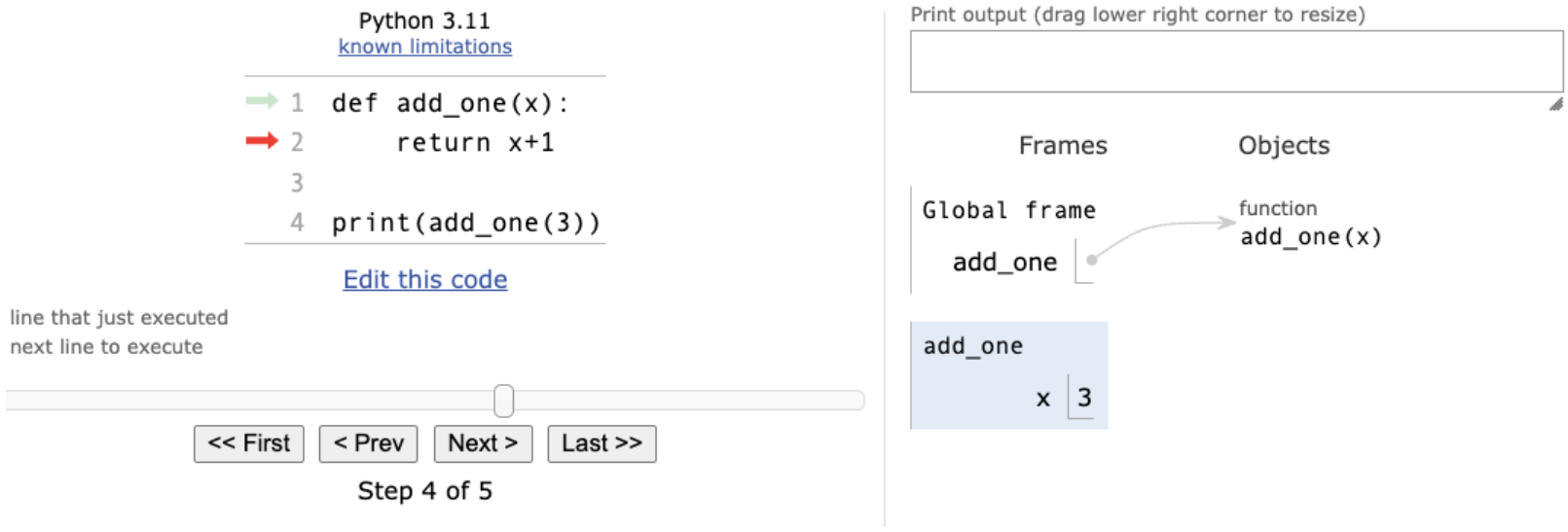
Referencing a function

- Functions in Python are objects, just like everything else!

```
>>> y = add_one
>>> y
<function add_one at 0x108e962f0>
>>> y(2)
3
>>> my_abs = abs
>>> my_abs(-10)
10
```

- What the keyword `def some_function` actually does is create a new function object and reference it with `some_function` .

Referencing a function



https://pythontutor.com/render.html#mode=display

Higher-order functions

- We can pass functions as parameters, return them from functions, even create them on the fly!
- Such functions are known as **higher-order functions**.
- We can define our own or work with built-in higher-order functions.

Take a look at `higher_order_functions.py`

- What do the first four functions do?
  - Take two parameter and do standard mathematical calculations.
- What does *add2* do in `higher_order_functions.py` ?
  - Takes one parameter, a tuple of two items.
  - Unpacks the tuple, adds and returns its items.
- What does `double` do in `higher_order_functions.py`?
  - Takes one parameter.
  - Multiplies by 2 and returns it.
- What does `is_even` do in `higher_order_functions.py` ?
  - Takes one parameter, a number.
  - Returns whether this number is even.

Take a look at `higher_order_functions.py`

- What does `apply_function` do?
  - Takes three parameters
  - The first is a function! Then passes second and third parameter to that function and returns the result.

```
>>> apply_function(add, 2, 3)
5
>>> apply_function(subtract, 2, 3)
-1
```

Take a look at `higher_order_functions.py`

- What does the `apply_function_to_list` function?
  - Takes a function `f` and a list `some_list` as parameters
  - Creates an empty list `result`
  - Iterates through each value in `some_list`
  - Applies the function `f` to each value and appends it to the `result`.
- High-level: applies the function `f` to each element in `some_list` and returns a new list containing the result from each of those applications.
- For example:

```
>>> apply_function_to_list(double, [1, 2, 3, 4])
[2, 4, 6, 8]
>>> apply_function_to_list(add2, [(1, 2), (3, 4)])
[3, 7]
```

Take a look at `higher_order_functions.py`

- What does the `apply_function_to_tuple` function do?
  - Takes a function `f` and a list `some_list` of 2-tuples as parameters
  - Iterates through each 2-tuple in `some_list` and unpacks it
  - Applies the function `f` on the two unpacked items
  - Appends the result of this application to a list that is returned at the end.
- For example:

```
>>> apply_function_to_tuple(add, [(1, 2), (3, 4)])
[3, 7]
```

map

- `apply_function_to_list` is actually built-in to Python and is called `map`.
- `map` takes as input a function and something that is iterable
  - only difference from apply_function_to_list is that it returns a map object (not a list), which is also iterable. For example:

```
>>> map(double, [1, 2, 3, 4])
<map object at 0x7f7ff809b128>
>>> for val in map(double, [1, 2, 3, 4]):
        print(val)

2

4

6

8
```

map

- By itself, this may not seem useful, but we can do more complicated things. What would this print?

```
for val in map(double, map(double, [1, 2, 3, 4])):
    print(val)
```

```
4
8
12
16
```

The first call to `map` doubles it and then we iterate on this result and double it again!

Take a look at `higher_order_functions.py`

- What does the `filter_list` function do?
- What does `some_function` return when passed a `some_list` element?
  - It should return a `bool`
- Similarly to *map*, Python has a built-in function for this behavior called `filter`.
- The `filter` function returns a list of all elements of `some_list` that would return `True` when passed to `some_function`. Note how it differs from *map*. For example,

```
>>> list(map(is_even, [1, 2, 3, 4]))
[False, True, False, True]
>>> list(filter(is_even, [1, 2, 3, 4]))
[2, 4]
```

Anonymous functions

- It can be a bit annoying having to define all of these simple functions to simply pass them as an argument to another function.
- Python allows us to create **anonymous functions**, i.e., functions that don't have an explicit name, but are simply code.
- The syntax is:
  `lambda <input>: <expression>`
- <input> is the parameter(s) to the anonymous function.
  - If you need to pass multiple inputs, just pass them as a tuple.
- <expression> is the body of the function that is executed and returned. It can only be a single expression (i.e., something that represents a value).

Anonymous functions

- An example:

```
>>> lambda x: x+1
<function <lambda> at 0x7f7ff80981e0>
```

- Notice that it gives the same function type back, but it doesn't have a name!

```
>>> (lambda x: x+1)(2)
3
```

- We can also associate it with a variable and call it, e.g.,

```
f = lambda x: x+1
>>> f(2)
3
```

Anonymous functions

- An example:

```
>>> filter_list(lambda num: num % 2 == 0, [1, 2, 3, 4])
[2, 4]
```

Practice time

- Write anonymous functions for the following procedures:
- Multiply argument `num1` with argument `num2` and return the result:
- Add arguments `num1`, `num2`, `num3` and return the result

Answer

- Write anonymous functions for the following procedures:
- Multiply argument num1 with argument num2 and return the result:
  - `lambda num1, num2 : num1 * num2`
- Add arguments num1, num2, num3 and return the result
- `lambda num1, num2, num3 : num1 + num2 + num3`

`sorted` function

- Python supports a built-in function that takes an iterable object, such as a list, string, tuple, dictionary, etc), and returns a **new** sorted list from it without modifying the original.
- ```
  numbers = [3, 2, 5, 1]
  sorted_numbers = sorted(numbers)
  print(numbers) # remains [3, 2, 5, 1]
  print(sorted_numbers) # is now [1, 2, 3, 5]
  ```
- It can also sort reversely, e.g.,

```
sorted_numbers = sorted(numbers, reverse = True)
print(sorted_numbers) # is now [5, 3, 2, 1]
```
```

`sorted` function

- Sometimes, we want to sort our data in a non-standard way, e.g., instead of lexicographic order for a list of strings sort by length.
- sorted can take a key that is a function to be called on each list element prior to making comparisons.
- ```
  fruit = ['apple', 'banana', 'plum', 'kiwi']
  sorted_fruit = sorted(fruit, key = len)

  # secretely converts it into [5, 6, 4, 4] which uses to sort
print(sorted_fruit) # ['plum', 'kiwi', 'apple', 'banana']
```
```

Anonymous functions and `sorted`

- We can combine what we learned about anonymous functions and the `sorted` function.
- ```
  professors = [('Papoutsaki', 222), ('Kauchak', 224), ('Osborn', 103)]
  sorted_professors_by_office = sorted(professors, key = lambda x:x[1])
  # [('Osborn', 103), ('Papoutsaki', 222), ('Kauchak', 224)]
  sorted_professors_by_name = sorted(professors, key = lambda x:x[0])
  # [[('Kauchak', 224), ('Osborn', 103), ('Papoutsaki', 222)]
  ```
  ```

Practice time

- What would the following code do?

```
my_list = [3, 6, 3, 2, 4, 8, 23]
sorted(mylist, key=lambda x: x % 2 == 0)
```

https://stackoverflow.com/questions/8966538/syntax-behind-sortedkey-lambda

Answer

```
my_list = [3, 6, 3, 2, 4, 8, 23]

my_sorted_list = sorted(mylist, key=lambda x: x % 2 == 0)

# secretely considers it as [False, True, False, True, True, True,
False]

# Remember False==0, True == 1, thus False<True

# my_sorted_list == [3, 3, 23, 6, 2, 4, 8]
```

https://stackoverflow.com/questions/8966538/syntax-behind-sortedkey-lambda

Last thing on anonymous functions

- Let's look at this unusual function that inside it defines a function and returns it??

```
def kinda_crazy(num):
    def multiplier(x):
        return num * x
    return multiplier
>>>type(kinda_crazy(3))
<class 'function'>
>>>kinda_crazy(3)(2)
6
```

- We could use an anonymous function to be even more concise!

```
def crazy(num):
    return lambda x: num * x
>>> crazy(3)(2)
6
```

Code:

- higher_order_functions.py

Monte Carlo sampling

An approach to estimate values/probabilities by repeated random sampling

It’s a powerful technique used when it is difficult to solve the answer directly

For this class: a fun application higher-order functions

Monte Carlo sampling code to estimate area

```
import random


def monte_carlo(trials, test):
    count = 0

    for _ in range(trials):
        x = random.random()
        y = random.random()

    if test(x, y):
        count += 1

    return count / trials


def in_triangle(x,y):
return x + y < 1


def in_circle(x,y):
return x**2 + y**2 < 1
```

generates random points in x = (0-1) and y = (0-1)