Discrete Structures: Programming Constructs CMPSC 102

Oliver BONHAM-CARTER

Fall 2022
Week 4
Slides 01

Let's Discuss

Key Questions

How do I use **non-recursive** functions, **recursive** functions, and **lambda expressions** to perform mathematical operations such as computing the **absolute value** of a number and the **means** of a sequence of numbers?

Learning Objectives

To **remember** and **understand** some discrete mathematics and Python programming concepts, setting the stage for exploring of discrete structures.

Python Programming Retrospective

- Python code is to be **intuitive**
- Key components of Python programming include:
  - Function and their definitions
  - Input parameters for functions
  - The code block that completes the function's work
  - Return statements
  - Invocations of functions (*calls to functions*)
  - Collecting the returned values (function outpus).
- Investigate the ways to make the above commands possible with definitions and call using Python.

Values that Cannot be Negative

Self-Transcendence 3100 Mile Race

- Race around a New York Block
- Participants have 52 days to run 3,100 miles (4,989 km), meaning they must average 59.6 miles every day.
- *It is a race so long that runners need a haircut during it. They can get through 20 pairs of shoes. They run more than two marathons a day. For almost two months. On five hours of sleep a night.*[1]

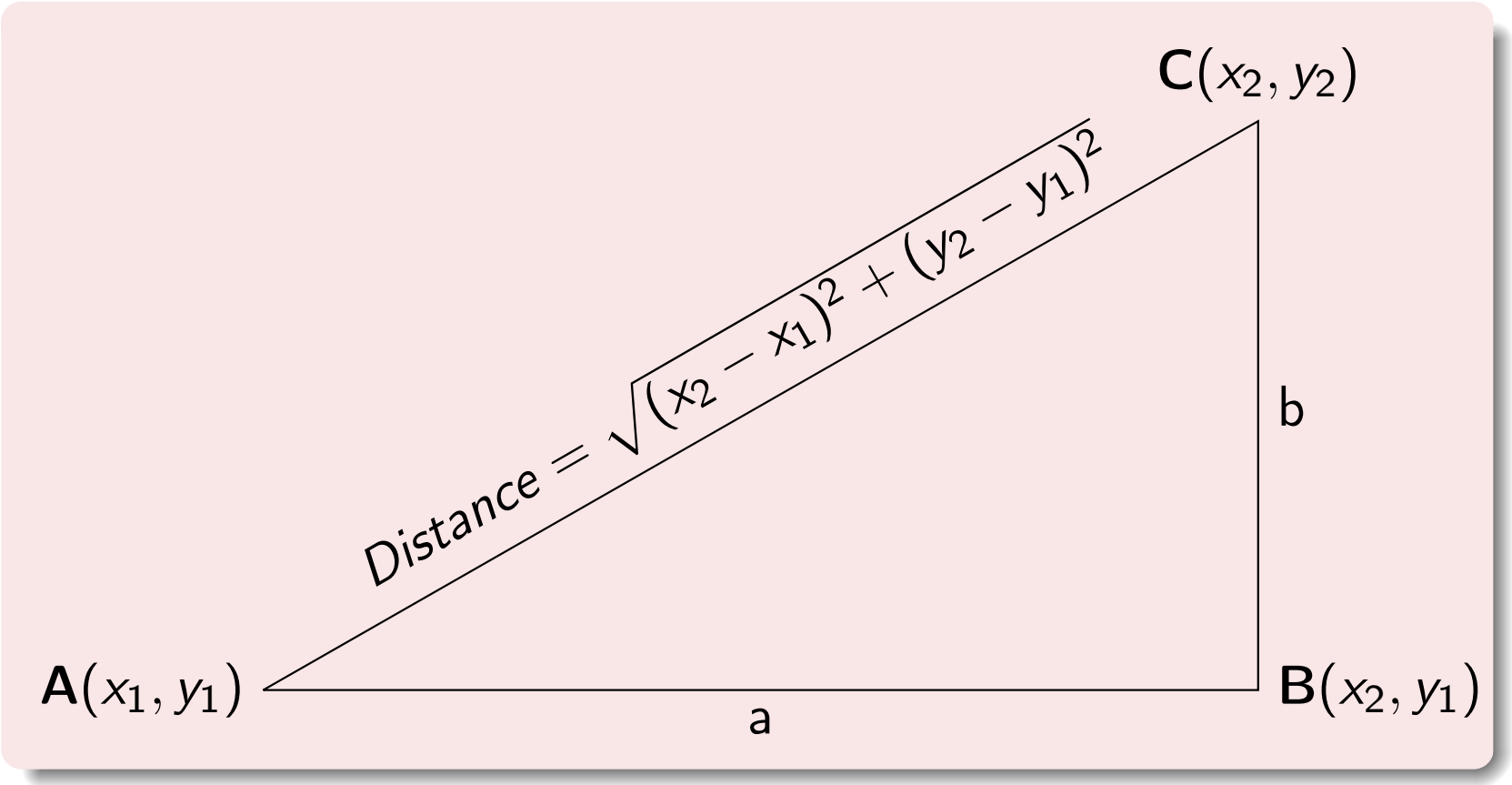
1 `https://www.bbc.com/sport/48702452`

2 `https://www.bangkokpost.com/world/2200371/worlds-longest-race-3-100-miles-around-a-new-york-block`

The course

Values that Cannot be Negative

Derived from the Pythagorean Theorem



- No negative *Distance* values
  - **By theory** (*can you run a negative distance!?*)
  - **By mathematics**: the square root function does not support negative values!

Absolute Value of a Number

Built in to Python

Speaking Pythonically

```
abs(x, /)
    Return the absolute value of the argument.
```

For example ...

```
>>>abs(-6)
6
>>>abs(-3 * 8)
24
```

- The absolute value of a number is its distance from zero

Go Live!

Try this on Live!

Function

```
#define a function to calculate abs val using cond logic
def abs(n: int) -> int:
    if n >= 0:
        return n
    return -n
# end of abs()
```

Function calls

```
# call the abs function with diff vals,  display the output
abs_positive_input = abs(100)
abs_negative_input = abs(-100)
abs_zero_input = abs(0)
print(f"The abs value of a pos num: {abs_positive_input}")
print(f"The abs value of a neg num:  {abs_negative_input}")
print(f"The abs value of zero is {abs_zero_input}")
```

Absolute Value of a Number

A function to calculate value

Function for finding absolute values

```
def abs(n):
    if n >= 0:
        return n
    else:
        return -n
```

Speaking Pythonically

- What is the meaning of the operator $>=$?
- What is the output of `print(str(abs(10)))` ?
- What is the output of `print(str(abs(-10)))` ?
- Are there other ways to implement this function ?

Another function to calculate value

Function for finding absolute values

```
def abs(n):
    if n >= 0:
        return n
    return -n
```

Speaking Pythonically

- Are there other ways to implement this function ?
  - Perhaps using `pow()` and `sqrt()`?

Another function to calculate value

Function for finding absolute values

```
import math

def abs(n):
    """ Square-rootsof squared values give positive values """
    return(math.sqrt(n**2))

def driver(myVal):
    print(f" the abs value of {myVal} is {abs(myVal)}")

myVal = -1
driver(myVal)
```

Speaking Pythonically

- Are there other ways to implement this function ?
  - Perhaps using `strings` and `integers`?

Supporting Functions
sqrt()

Function for finding square root values

```
def sqrt(num: int, sigFig: float):
    guess = 1.0
    while abs(num - guess*guess) > sigFig:
        guess = guess - (guess*guess - num)/(2*guess)
    return guess
```

Speaking Pythonically

- Remember, the square-root function that could be used here !?
- What is the meaning of `num:int` and `tol:float` ?
- How could we test the `sqrt` function using Unittest or other such as Pytest ?

Another function to calculate value

Function for finding absolute values

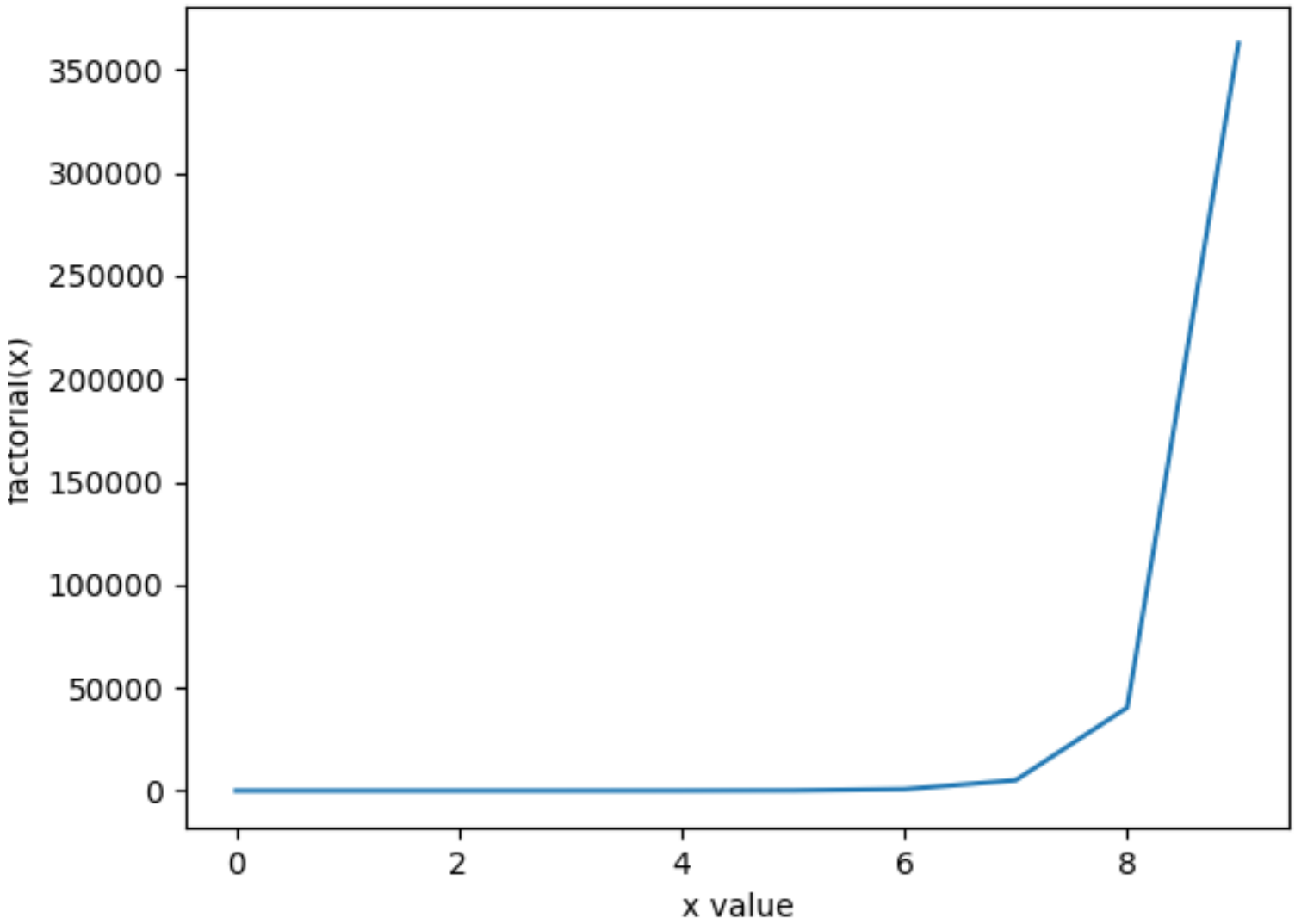
```
def abs_str(n_int):
    """ convert int to str, remove the "-", and return int."""
    n_str = str(n_int)
    return int(n_str[n_str.find("-")+1:])

def driver_str(myVal):
    print(f" the abs value of str {myVal} is {abs_str(myVal)}")

myVal = -10000
driver_str(myVal)
```

Speaking Pythonically

- Are there other ways to implement this function ?
  - There are many possibilities and efficiencies to choose!

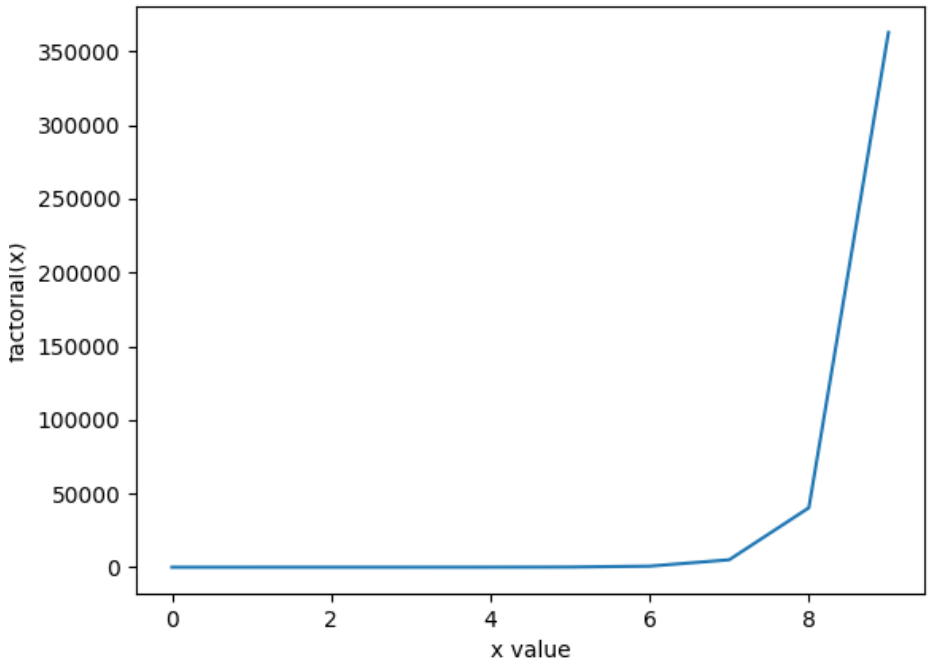
Factorials

Values get quickly get big



Factorials

Values get quickly get big



x	$fac(x)$
0	1
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800
11	39916800

Factorials

Factorials: one definition

$$N! = \prod_{i=1}^{N} i = 1 * 2 * .. * (N-1) * N$$

Factorials: another definition

$$N! = \frac{(N+1)!}{(N+1)} = \frac{(N+1)*N!}{(N+1)}$$

- Factorials are applied to integers

Factorials

Factorials

$N! = N * (N-1) * (N-2) * ... * (2) * (1)$

$5! = 5 * 4 * 3 * 2 * 1$

$4! = 4 * 3 * 2 * 1$

$3! = 3 * 2 * 1$

$2! = 2 * 1$

$1! = 1$

$0! = 1$ (Special case by convention)

Factorials defined

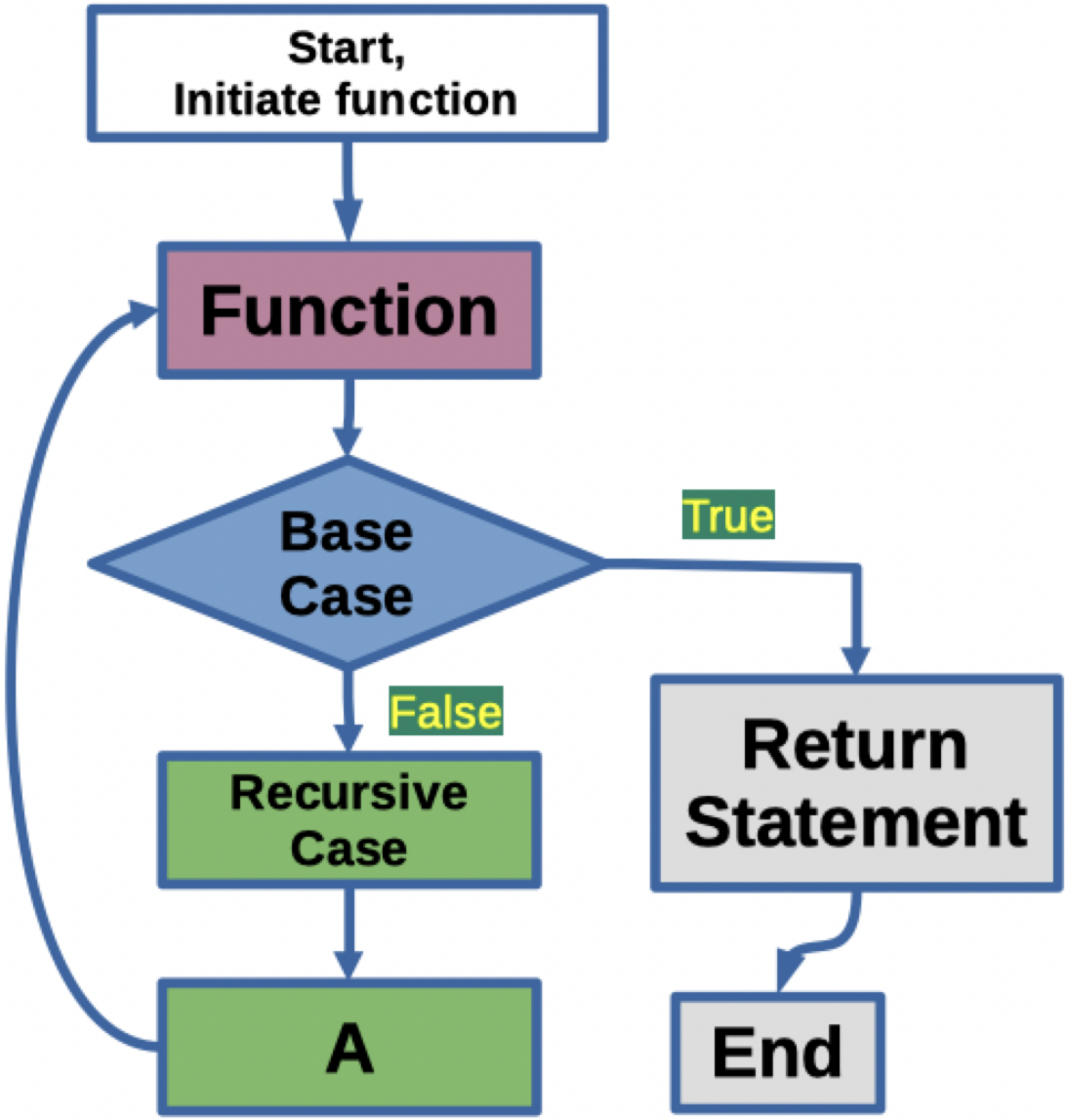
$N! = [\ (N-1)! + (N-2)!\ ] * (N-1)$

$7! = (6!\ + 5!\ )\ * 6$

$6! = (5! + 4!)\ * 5$

$5! = (4! + 3!)\ * 4$

Creating Solutions



Calculating Factorials by Recursion

```
def factorial(number: int):
    if number == 1:
        return 1
    return number * factorial(number - 1)
num = 5
print("The factorial of " + str(num)
+ " is " + str(factorial(num)))
```

- The recursive *factorial* function calls itself!
- How does this function ever stop executing?
- What are the benefits to using recursive functions?

The Code

```
def factorial(number: int):
    if number == 1:
        return 1
    return number * factorial(number - 1)

num = 5
print("The factorial of " + str(num) +
        " is " + str(factorial(num)))
```

- Where is the base case?
- Where is the recursive case?
- How does this function make progress to the base case?

Consider this...

ToDo

- Use the remaining time to construct code where we are using iteration to calculate a factorial.
- Please be sure to test your work to see what types of variables can cause it to crash.