Discrete Mathematics

Topic 01 : Computational Thinking

Lecture 01 : Fundamentals of Computation

Dr Kieran Murphy
Computing and Mathematics, SETU (Waterford).
(kieran.murphy@setu.ie)

Autumn Semester, 2023

Outline

- Using PyTutor with Colab
- Storing data and data types
- Making decisions
- Looping

Outline

Using PyTutor with Colab

Before we start covering Python we want to show you PyTutor in action. The following slides shows screenshots of the process but you should verify the steps yourself on your phone/tablet.

Step 1 — Click/Scan on QR Code

The following code outputs powers of 2, don't worry about the actual code, just make sure that you can open and use PyTutor . . .

```
1
1 powers = [0,1,2,3,4,5,6]
2 for p in powers:
3   print(p, 2**p)
```

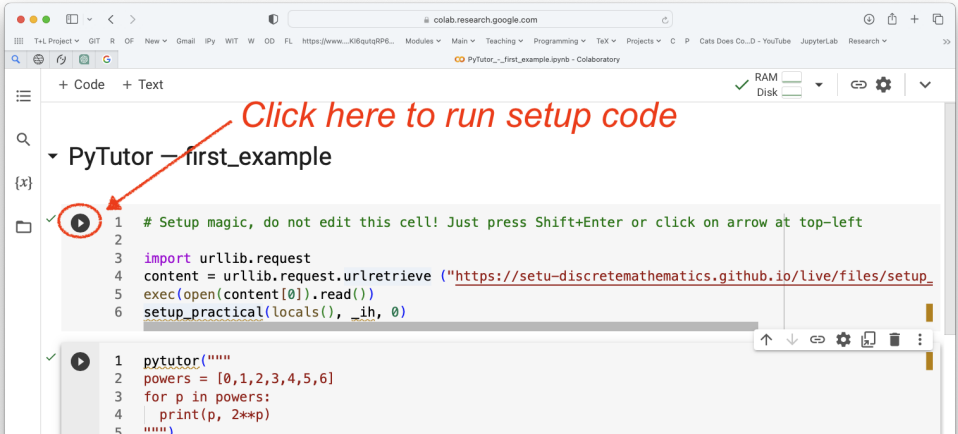
```
0 1
1 2
2 4
3 8
4 16
5 32
6 64
```

Using PyTutor with Colab II

This should open in Colab the following notebook.

Unlike our practical notebooks, don't bother clicking on File → Save a copy in Drive.

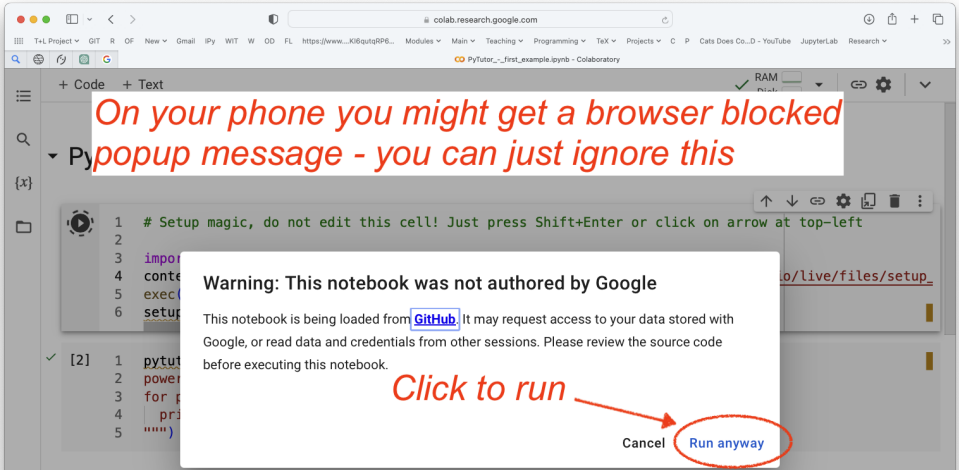
Step 2 — Execute the first cell to setup notebook.



Using PyTutor with Colab III

On executing the first cell you will get the following message. Click on Run anyway.

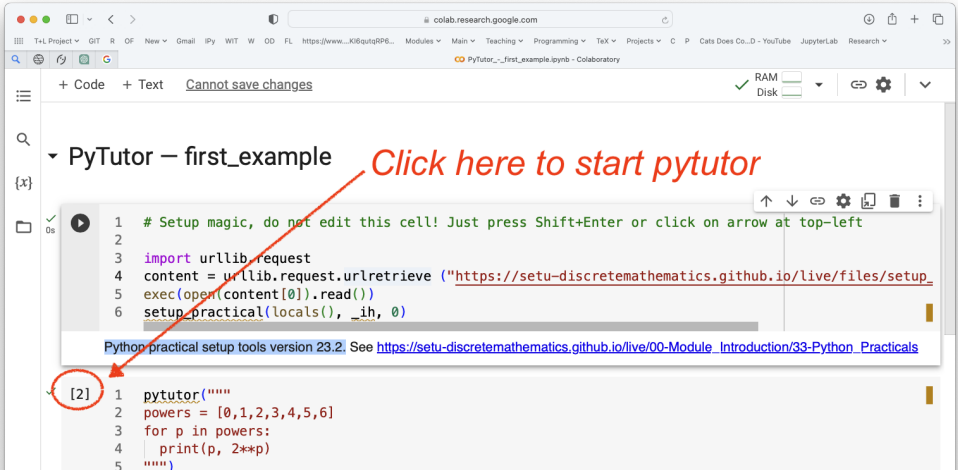
Step 3 — Click on Run anyway



Using PyTutor with Colab IV

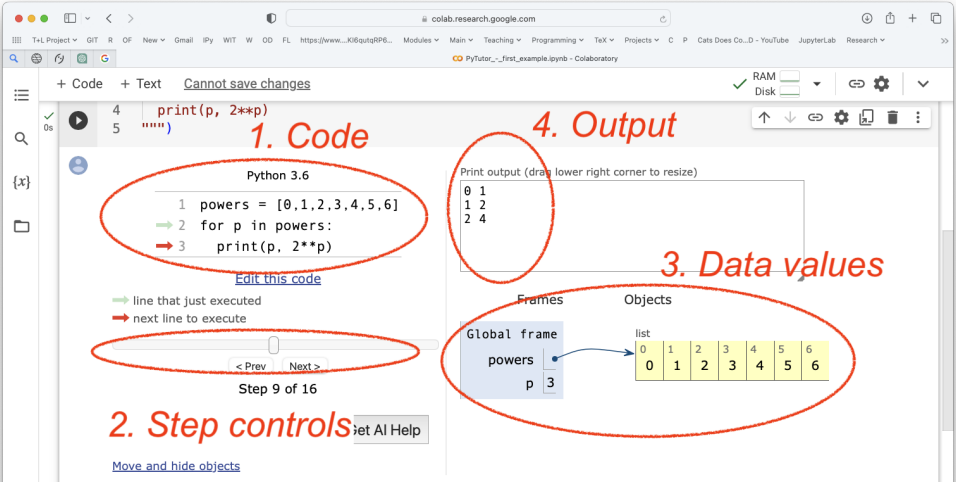
After executing the first cell you will get the usual "Python practical setup tools version 23.2".

Step 4 — Click on second cell to run code in PyTutor



Using PyTutor with Colab

You can now use PyTutor, to step back/forward through the code and see the current data values and resultinig output.



Outline

Brief History of Python

- Invented in the Netherlands, early 90s by Guido van Rossum.

  "Python is an experiment in how much freedom programmers need. Too much freedom and nobody can read another's code; too little and expressiveness is endangered."

  – Guido

- Named after Monty Python.
- Scalable, object oriented and functional from the beginning
- Python 3.0 was released in 2008, to rectify certain flaws in Python 2.*.
- Most popular language for machine learning and data mining.

Python's Benevolent Dictator For Life

First Look at Python Code

To get an idea of Python, we will take a small piece of code*

```
# Solution to Euler problem 2

# Calculate the sum of the even-values in the Fibonacci sequence
#    1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...
# value that do not exceed four million,

last = 1
current = 2

answer = 0
while current <= 4_000_000:
    if current % 2 == 0:
        answer += current
    last, current = current, last + current

print(answer)
```

*This is a solution to the Euler Problem 2, at the programming competition site, projecteuler.net.

First Look at Python Code

To get an idea of Python, we will take a small piece of code*

```
1  # Solution to Euler problem
2
3  # Calculate the sum of the ev
4  #    1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...
5  # value that do not exceed four million,
6
7  last = 1
8  current = 2
9
10 answer = 0
11 while current <= 4_000_000:
12     if current % 2 == 0:
13         answer += current
14     last, current = current, last + current
15
16 print(answer)
```

The character # indicates an end of line comment. In each line everything after the # is ignored by the computer

*This is a solution to the Euler Problem 2, at the programming competition site, projecteuler.net.

First Look at Python Code

To get an idea of Python, we will take a small piece of code*

```
# Solution to Euler problem 2

# Calculate the sum of the
#    1, 2, 3, 5, 8, 13, 21,
# value that do not exceed

last = 1
current = 2

answer = 0
while current <= 4_000_000:
    if current % 2 == 0:
        answer += current
    last, current = current, last + current

print(answer)
```

Use = to store data.

On the left of =, we have the **identifier** name(s)
On the right of =, we have the data value(s)

Unlike other languages (e.g., Processing) we don't need to state the data type. (More on this later.)

*This is a solution to the Euler Problem 2, at the programming competition site, projecteuler.net.

First Look at Python Code

To get an idea of Python, w

```
5
1  # Solution to Euler proble
2
3  # Calculate the sum of the
4  #    1, 2, 3, 5, 8, 13, 21,
5  # value that do not exceed
6
7  last = 1
8  current = 2
9
10 answer = 0
11 while current <= 4_000_000:
12     if current % 2 == 0:
13         answer += current
14     last, current = current, last + current
15
16 print(answer)
```

A core feature of Python is **indent**ing

indent is the spacing at **start** of Python lines of code. It is used to specify blocks of code, for functions, loops or decisions.

Here we have a `while` loop block with lines 12–14. Inside that, we have an `if` decision block with line 13.

Note the `:` at end of line **before** code block

Other languages (e.g., Processing) use brackets `{` and `}` to specify blocks. Python doesn't and this results in cleaner code.

*This is a solution to the Euler Problem 2, at the programming competition site, projecteuler.net.

First Look at Python Code

To get an idea of Python, we will take a small piece of code*

```
1  # Solution to Euler problem 2
2
3  # Calculate the sum of the
4  #    1, 2, 3, 5, 8, 13, 21,
5  # value that do not exceed
6
7  last = 1
8  current = 2
9
10 answer = 0
11 while current <= 4_000_000:
12     if current % 2 == 0:
13         answer += current
14     last, current = current, last + current
15
16 print(answer)
```

Python has lots of little features that make coding nicer.

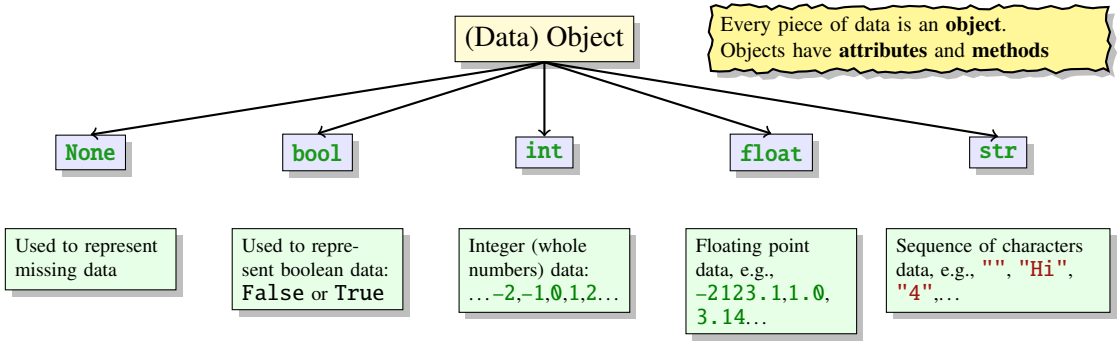
For example:

- We can use underscore _ to represent thousand separator in numbers (line 11).
- We can assign multiple values at the same time (line 14).

*This is a solution to the Euler Problem 2, at the programming competition site, projecteuler.net.

Data and Data Types

Python has 5 main primitive data types:



- An **Object** stores data in its **attributes**, and **methods** are used to change an object.
- In Python, the type of the data is automatically determined (unlike Processing).
- The type determines what you are allowed to do to a piece of data.
- Function `type` will return the type of a piece of data.

Data and Data Types II

```
w = None

x = 4
y = "4"
z = 4.0

print(type(w), type(x), type(y), type(z))

x = x * 10
y = y * 10
z = z * 10

x = x * 1_000_000_000
z = z * 1_000_000_000

x = x / 1_000_000_000
z = z / 1_000_000_000

print(type(w), type(x), type(y), type(z))
```

Data and Data Types II

Python infers the type from the data or from the result of an operation on data.

```
8
1  w = None
2
3  x = 4
4  y = "4"
5  z = 4.0
6
7  print(type(w), type(x), type(y), type(z))
8
9  x = x * 10
10 y = y * 10
11 z = z * 10
12
13 x = x * 1_000_000_000
14 z = z * 1_000_000_000
15
16 x = x / 1_000_000_000
17 z = z / 1_000_000_000
18
19 print(type(w), type(x), type(y), type(z))
```

Data and Data Types II

Operations (here multiplication using $*$) can do different things based on the type.

```
9
1  w = None
2
3  x = 4
4  y = "4"
5  z = 4.0
6
7  print(type(w), type(x), type(y), type(z))
8
9  x = x * 10
10 y = y * 10
11 z = z * 10
12
13 x = x * 1_000_000_000
14 z = z * 1_000_000_000
15
16 x = x / 1_000_000_000
17 z = z / 1_000_000_000
18
19 print(type(w), type(x), type(y), type(z))
```

Data and Data Types II

Using function **type** on an object is a common feature of Python programming.

```
10
1  w = None
2
3  x = 4
4  y = "4"
5  z = 4.0
6
7  print(type(w), type(x), type(y), type(z))
8
9  x = x * 10
10 y = y * 10
11 z = z * 10
12
13 x = x * 1_000_000_000
14 z = z * 1_000_000_000
15
16 x = x / 1_000_000_000
17 z = z / 1_000_000_000
18
19 print(type(w), type(x), type(y), type(z))
```

Collections: `set`, `list`

We will cover collections in more detail later, but for now we have:

Sets

- A `set` is collection of **distinct**, **unordered** values.
  - **distinct** means a set cannot hold the same piece of data more than once.
  - **unordered** means we cannot sort the elements of a set of ask "what element is first?" etc.
  - We can manipulate sets using union `|`, intersection `&`, set minus `-` operations.

Lists

- A `list` is collection of **ordered** values.
  - **ordered** means the values appear in a sequence (i.e. have position). So we can talk about which value appear before/after another value. (**ordered** $\neq$ **sorted**)
  - Data values do not have to be distinct.
  - The position of a data value Python is called its **index**. Since Python is a **zero-based language**, the position starts at 0.
  - Lists are a BIG DEAL in python and we have many operations to manipulate them (more later).

Collections: set

```
11
1  z = set() # creating a empty set
2
3  # defining sets by stating values
4  a = {1,3,1,2,1,5,4}
5  b = {1,3}
6
7  print(len(a)) # size of set
8
9  c = a & b # intersection
10 print(c)
11
12 c = a | b # union
13 print(c)
14
15 c = a - b # set difference
16 print(c)
17
18 c = b - a # set difference
19 print(c)
```

Collections: `list`

```
12
1  z = []  # creating a empty list
2
3  # defining lists by stating values
4  a = [1,3,1,2,1,5,4]
5  b = [1,3]
6
7  print(len(a)) # size of list
8
9  c = a + b # appending lists
10 print(c)
11
12 value = c[2] # list indexing ZERO-BASED
13 print(value)
14
15 d = c[2:5] # slicing SEMI-OPEN notation
16 print(d)
17
18 value = c[-4] # negative indexing
19 print(value)
```

Looping: `for`, `while`

`for` — Looping when you know how many times you want to repeat

- •

`while` — Looping when you don't know how many times you want to repeat

- •

Making Decisions: `if`, `elif`, `else`

Functions: `def`, `return`