Introduction to Python

Week 3 - File input/output, lambda, sort, filter, algorithms

Turing Students
ROTTERDAM

File I/O

Turing Students ROTTERDAM

- Files you see in File Explorer/Finder
  - Referred to by a path
    - C:\Users\yoyo\Documents\example.csv
    - /Users/yoyo/Documents/example.csv
    - /home/yoyo/Documents/example.csv
- Can be opened, read, written and closed
  - Closing is important, or it gets locked and unavailable to be written by other software

Opening a file

- `open(path, mode)`
  - returns file object
- `with` statement
  - Closes the file after running code inside

Access modes [1] [2] [3]

`r`	reading (default)
`w`	writing, truncating the file first
`x`	exclusive creation, fails if file exists
`a`	writing, appending to the end if file exists
`b`	binary mode
`t`	text mode (default)
`+`	updating (reading and writing)

Turing Students ROTTERDAM

```
# Create a file if it doesn't exist
f = open("hello.txt","w")
# Write something
f.write("Hello world!")
# Close
f.close()

with open("hello.txt", "r") as f:
    # Print the file contents
    print(f.read())
# File is closed automatically
```

Reading from a file

- `fileobj.read(size)`
  - Returns `size` bytes of a file
  - Returns the whole file if `size` is not given
- `fileobj.readline()`
  - Reads a line
- `fileobj.readlines()`
  - Return a list of lines of the file
  - Is same as the for-loop to the right
- Note: file pointer (caret/cursor) moves as you read, and it doesn't go back

Turing Students ROTTERDAM

```
# Read whole file
with open("hello.txt", "r") as f:
    print(f.read())

# Read one line
with open("hello.txt", "r") as f:
    print(f.readline())

# Read line by line
with open("hello.txt", "r") as f:
    for line in f:
        print(line)
```

Writing to a file

- `fileobj.write(string)`
  - Returns the number of characters written
- `fileobj.writelines(lines)`
  - Write a list of lines
  - Line separator not included

Turing Students ROTTERDAM

```
with open("hello.txt", "a") as f:
    # Write a string
    f.write("Hello world!\n")
    # Write a list of items
    lines = ["apple\n", "orange\n",
"banana\n"]
    f.writelines(lines)
```

File I/O (6)

Turing Students ROTTERDAM

Same example but when using ‘with’, it automatically close & return

Hence, it makes life easier

```
# 'with' calls close() automatically
with open('output2.txt','w') as f:
    f.write('hello world \n')
    f.write('Python is funn')
```

File I/O (8)

Let’s read output3.txt

```
# read output3.txt
with open('output3.txt', 'r') as f:
    print('----')
    print(f.read())

# read one line
with open('output3.txt', 'r') as f:
    print(f.readline())

# it seperates file by lines and stores in a list
with open('output3.txt', 'r') as f:
    print(f.readlines())
```

File I/O (9)

Turing Students ROTTERDAM

Last exercise!

- sales.txt contains the sale
- Read the file and calculate total & average of the sales
- Then print them in another file

File I/O (10)

Turing Students ROTTERDAM

```
infile = open('dataSet/sales.txt','r')
outfile = open('dataSet/salesoutput.txt','w')
summ = 0
count = 0
for line in infile:
    summ += int(line)
    count += 1
outfile.write('sum is '+ str(summ))
outfile.write('\n average is '+ str(summ/count))
infile.close()
outfile.close()
```

Lambda functions

`lambda` gives a special way to write functions in Python. They are used for writing anonymous functions in one line instead of giving a name.

An anonymous function is often used when you need to use a function once, and you are not referring to it again. The following examples will demonstrates those situations.

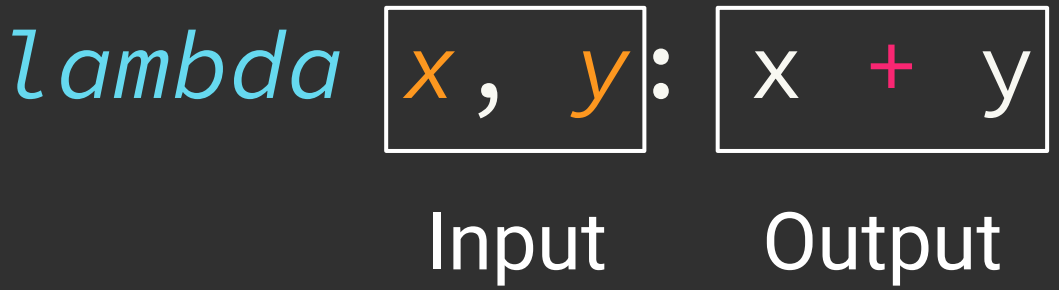
Turing Students ROTTERDAM

```
# Normal function definition
def add5(x):
    return x + 5

print(add5(5))

# This is the same as above
add5 = lambda x: x + 5
print(add5(5))
```

Lambda function: Syntax



Lambda functions

`lambda` functions are similar to a normal function, except a lambda function can only contain one line of code, and the output of that statement is returned.

Compared with typical functions with `def`, a lambda function looks more concise, and is often used for simple transformations.

Turing Students ROTTERDAM

```
add5 = lambda x: x + 5
print(add5(5))

# It also accepts multiple
parameters
add = lambda x, y: x + y
print(add(2, 3))


# Let's define add5 based on add
add5 = lambda x: add(x, 5)
print(add5(5))
```

Lambda functions

Another rarer use case of lambda functions is to create functions out of functions.

This example is just to show the expressiveness of anonymous functions, practically speaking you won't be using it like this.

Lambda functions are based on lambda calculus introduced by Alonzo Church, which finds its way into functional programming languages such as Haskell, OCaml, Reason, F#, etc.

Turing Students ROTTERDAM

```
add = lambda x, y: x + y
print(add(2, 3))

addx = lambda x: lambda y: x + y
add5 = addx(5)
print(add5(5))

# Let's look at addx again
def addx(x):
  return lambda y: x + y
```

Sort function

Turing Students ROTTERDAM

- `list.sort()`
  - builtin function to sort
  - only defined for lists
  - Reorders items in the same list
- `sorted()`
  - builtin function that builds a **new** sorted list from an iterable
  - accepts any iterable

Sort function (2)

Turing Students ROTTERDAM

list.sort()

```
numbers = [1, 3, 4, 2, 5]

'''by default,
sorting list of Integers
in ascending order'''
numbers.sort()
print(numbers)
```

```
[1, 2, 3, 4, 5]
```

```
# sort by descending order
numbers.sort(reverse=True)
print(numbers)

# sort by ascending order
numbers.sort(reverse=False)
print(numbers)
```

```
[5, 4, 3, 2, 1]
[1, 2, 3, 4, 5]
```

Sort function (3)

Turing Students ROTTERDAM

sorted()

```
print(sorted(numbers))
print(sorted({1: 'D', 2: 'B', 3: 'B', 4: 'E', 5: 'A'}))
print(sorted({'q':1, 'w':2, 'e':3, 'r':4, 't':5, 'y':6}))
```

```
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
['e', 'q', 'r', 't', 'w', 'y']
```

Filter

Turing Students ROTTERDAM

- You can filter the given sequence using the function
- filter(function, sequence)
- Filter function resembles a for loop but it is a built-in function and faster

Filter (2)

```python
# function that filters vowels
def fun(variable):
    letters = ['a', 'e', 'i', 'o', 'u']
    if (variable in letters):
        return True
    else:
        return False

# sequence
sequence = ['g', 'e', 'e', 'j', 'k', 's', 'p', 'r']

# using filter function
filtered = filter(fun, sequence)

print('The filtered letters are:')
for s in filtered:
    print(s)
```

```
The filtered letters are:
e
e
```

Turing Students ROTTERDAM

Filter (3)

Turing Students ROTTERDAM

When ‘filter’ meets ‘lambda’

```
filter(lambda x: x < 5, range(10))
```

```
<filter object at 0x000001FC8C218828>
```

```
# pile them in the list
list(filter(lambda x: x < 5, range(10)))
```

```
[0, 1, 2, 3, 4]
```

Algorithms: Sorting

Turing Students ROTTERDAM

Given a list of unique numbers, how do you sort them from smallest to largest?

Input	[9, 5, 1, 3, 4, 7, 8, 5, 2]
Output	[1, 2, 3, 4, 5, 6, 7, 8, 9]

Insertion sort

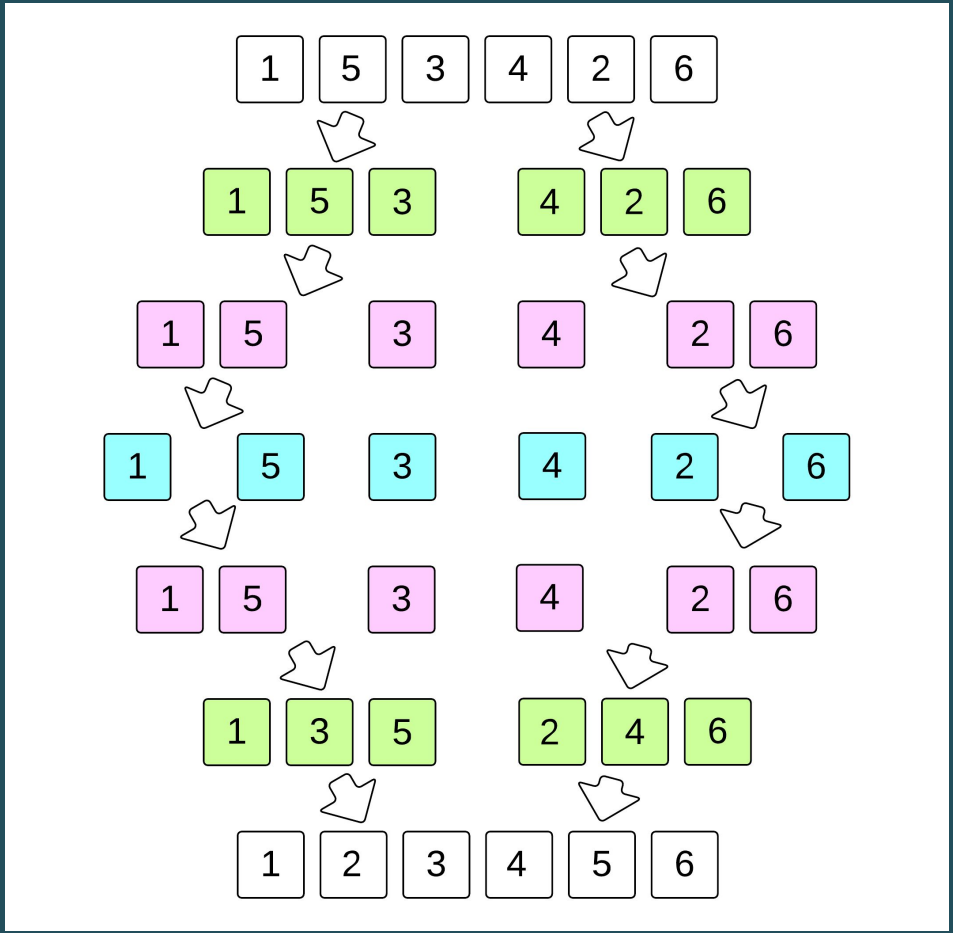
Insertion sort is an in-place sorting algorithm that sorts items one item at a time, by removing one item, finding the location it belongs in the sorted list, and put there. The process is repeated until there are no items left.

```
def insertion_sort(a):
    i = 1
    while i < len(a):
        j = i
        while j > 0 and a[j - 1] > a[j]:
            a[j - 1], a[j] = a[j], a[j - 1]
            j = j - 1
        i = i + 1

test = [5, 4, 3, 2, 1]
insertion_sort(test)
print(test)
```

Turing Students ROTTERDAM

Merge sort



Turing Students ROTTERDAM

Big-O notation

Turing Students ROTTERDAM

The Big-O notation describes the performance of an algorithm by the magnitude of its operations.

	Best	Average	Worst
Insertion sort	O(n)	$O(n^2)$	$O(n^2)$
Mergesort	O(n log n)	O(n log n)	O(n log n)
Timsort	O(n)	O(n log n)	O(n log n)