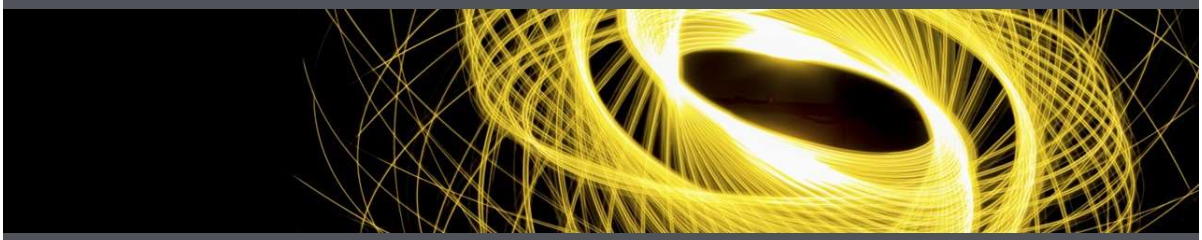
Department of Computer Science

University of Reading

CSMAD21 – Applied Data Science with Python



Python for Data Science - Data Types, Operators and Flow Control

1

University of Reading

Lecture Objectives

- Differentiate, interact, implement and modify the datatypes available in Python.
- Identify, configure and apply flow control structures and comparison operators.

2

University of Reading

Outline

- **Datatypes:**
  - Numbers
  - Strings
  - Boolean
  - Tuples
  - Sets
  - List
  - Dictionaries
- **Flow Control Structures**
  - If-Else
  - For loops
  - While
- **Comparison Operators**
  - And
  - Or

3

University of Reading

Basic Data Types – General Concepts

- To differentiate the data types that python provides, first we need to understand some general concepts:
  - Ordered: The values are ordered.
  - Mutable / Immutable : The values can/cannot be altered after creation.
  - Nestable: Can contain its same datatype.

4

University of Reading

Basic Data Types - Overview

- Python includes all the built-in data types and operations that you would expect in a programming language:

```
x = 1
y = 2.5
fname = 'Mike'
sname = 'Jones'
```

- **Note the lack of type declarations!**
- We will see later that everything is an object in Python.

5

In [ ]:
```
#Integer and floating point
x = 1
y = 2.5
#Booleans:
yes = True
no = False
#Strings
fstname = "Mike"
srname = "Jones"
#List:
a = [2,3,2,3,3]
#Tuples
t = (1,2,3)
#Dictionary:
d = { 1: "Hello", 2: "world" }
```

In [ ]:
```
d?
```

Basic Datatypes - Printing

In [ ]:
```
print('Integer:')
print(x)
###
print('Float point:')
print(y)
##
print('Boolean:')
print(yes, no)
##
print('String:')
print(fstname, srname)
##List
print('List:')
print(a)
##Tuple
print('Tuple:')
print(t)
```

```
##Dictionary
print('Dictionary:')
print(d)
```

In [ ]:
```
##Combined print:
print('My name is {one} {two} and I have {three} dog.'.format(one=fstname,two=srname
print('My name is {} {} and I have {} dog.'.format(fstname,srname,x))
print('My name is',fstname + ' ' + srname, 'and I have',x,'dog.')
```

Basic Datatypes - Basic Operations - Numbers

In [ ]:
```
##Addition, Substraction, Multiplication and Divisions
print('Addition:',5+5,'\nSubstraction:',5-5,'\nMiltiplication:', 5*5,'\nDivission:',
```

In [ ]:
```
4+1
```

In [ ]:
```
##Power, Integer divide operation and modulo operator
print('Power:',5**3,'\nInteger divide:',7//3,'\nModulo Operator:', 7%3)
```

University of Reading

Working with Lists

- In Python lists are collections of objects that are:
  - Ordered
  - Mutable
  - Have variable length
  - Can store multiple types
  - Can be nested
- In other words objects are stored in the list in an order, and the list can be modified. Lists can be made longer or shorter, and can store combinations of different types of object in a single list, including other lists.
- For example, these are all valid lists:

6

In [ ]:
```
l1 = [3,2,1]
l2 = ["3",2,"one"]
l3 = [[3,4,5,6],2,"one"]
```

In [ ]:
```
l3
```

- Lists are accessed **using zero based indexing**:

In [ ]:

```
l1[2]
```

- As we have seen lists can be initialised using the syntax. It is also possible to create an empty list using [].

In [ ]:
```
b = []
b
```

- We can find the length of a list using **len**:

In [ ]:
```
len(l3)
```

Concatenate Lists

- Lists can be concatenated using **+**:

In [ ]:
```
l1 + l2
```

In [ ]:
```
l4 = l1+l2
l4
```

In [ ]:
```
l4 =  l4 + ['two', ['dog', 'cat']]
l4
```

In [ ]:
```
len(l4)
```

Modify Lists

- Lists are mutable so they can be modified after they have been created:

In [ ]:
```
a = [1,2,3,4,5,6]
a
```

In [ ]:
```
a[3] = 7012
a
```

- Assignments are bounds checked so attempting to index a list outside of it bounds will produce a runtime error.
- Note that for a list a with x=len(a), the valid indices range from 0,1,...,x−1.

In [ ]:
```
len(a)
```

In [ ]:
```
a[6] = 7
```

University of Reading

Working with Lists - Slicing

- It is also possible to select slices of lists, that return a range within the list:

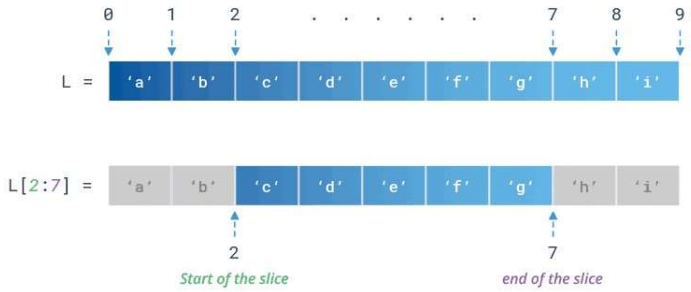


Figure 1: List Slicing (Learn by Example, 2019)

7

In [ ]:
```
##Example:
a = [1,2,3,4,5,6]
a
```

In [ ]:
```
a[2:5]
```

- Omitting an index in a slice ranges over the whole of the rest of the list:

In [ ]:
```
a[4:]
```

- You can also use negative indices to index from the end of a list:

In [ ]:
```
a[-3]
```

In [ ]:
```
a[:-2]
```

In [ ]:
```
a[-5:]
```

In [ ]:
```
a[:-2]
```

In [ ]:
```
a[1:-1]
```

Other operations:

- Repeating:
- Iterating
- Membership test

In [ ]:
```
#Repeating
a = [1,2,3,4,5,6]
#a
a*4
```

In [ ]:
```
#Iterating
for x in a[0:3]:
    print(x)
```

In [ ]:
```
#Membership test
2 in a
```

Methods

- Lists in Python are objects that have methods, similarly to Java. For any list a we can apply:
  - Append: Append value x to list a.
  - Insert: insert value x into list a at index ix
  - Sort
  - Reverse

In [ ]:
```
a = [1,4,2,8,5,9,0]
a.append(200) # Append value x to list a a.insert(ix,x)  # Sort a a.reverse() # Reve
a
```

In [ ]:
```
# Insert value x into list a at index ix
a = [1,4,2,8,5,9,0]
a
a.insert(2,500) ## alternative to a[2] = 500
a
```

In [ ]:
```
a = [1,4,2,8,5,9,0]
a.sort()
a
```

In [ ]:
```
b = ['a','b','d','w','c']
b.sort()
b
```

In [ ]:
```
a.reverse()
a
```

- Lists in CPython (the most common Python interpreter) are implemented as dynamic arrays - arrays of pointers to Python objects, that are dynamically resized as needed.

- This means that accessing or setting list elements has time complexity of O(n). Although Python has built in operations like in, these still take the amount of time you would expect.

Where n is the length of the list.

In [ ]:
```
#O(n) as each list member needs to be checked.
5 in a
```

In [ ]:
```
#O(n) as the list needs to be shifted to make space.
a.insert(2,8)
a
```

In [ ]:
```
#O(n) as the list needs to be shifted to erase an entry.
del a[1]
a
```

University of Reading

Working with Strings

- Strings behave just like lists.
- We can find the first character of a string name using name[0], and it's length in characters using len(name).
- Examples with :
  - Concat
  - Slicing

8

In [ ]:
```
str1 = 'Hello '
str2 = 'how are '
str3 = 'you?'
#str3
## Concat
str1 + str2 + str3
```

In [ ]:
```
##Slicing
str4 = str1 + str2 + str3
#str4
len(str4)
```

In [ ]:
```
#str4[3]
str4[5:-2]
```

In [ ]:
```
##The methods of the lists are not available for strings.
str4.append('?')
#str4
```

In [ ]:
```
str4 = str4 + '?'
str4
```

University of Reading

Working with Dictionaries

- In Python dictionaries act as efficient lookup tables. They are:
  - Unordered
  - Accessed by keys
  - Mutable (they can be changed)
  - Able to grow and shrink dynamically
  - Nestable (a dictionary entry can be another dictionary, or anything else!)
- Each key has an associated value. Keys do not all have to have the same type, and values do not have to all have the same type.
- **Keys can only be immutable types, integers, strings or tuples.**

9

Dictionary initialisation

- An empty dictionary can be created using {}. Values are then assigned to keys using a similar syntax to lists:

In [ ]:
```
d = {}
d['Python'] = 'Snake' # Key 'Python' has value "Snake"
d[4] = 7 # Key 4 has value 7
d
```

- Dictionaries can also be initialised by assigning values to keys:

In [ ]:
```
d = { 'Python': 'Snake', 4: 7 }
d
```

Lookups

- Once a dictionary has been created, values corresponding to a key can be accessed:

In [ ]:
```
d['Python']
```

In [ ]:
```
d[4]
```

- **However since dictionaries aren't ordered, operations like concatenation and slicing are not available.**

In [ ]:
```
d[0:1]
```

Updating

- Dictionaries are **mutable** so they can be modified once they are created, and values associated with keys can be changed:

In [ ]:
```
d
```

In [ ]:
```
#d[4] = d[4] + 1
d['Python'] = 'Programming Language'
d
```

Methods

- Just like lists, dictionaries are objects with associated methods:

In [ ]:
```
d = { 'Python': 'Snake', 4: 7 }
d.keys() # ['Python',4]
#d
```

In [ ]:
```
d.values()
```

- **Note that there is no fixed ordering to the output of the keys and values methods.**
- **You should assume that they will be arbitrary and could change depending on the operating system, computer and Python version being used.**

Iterating

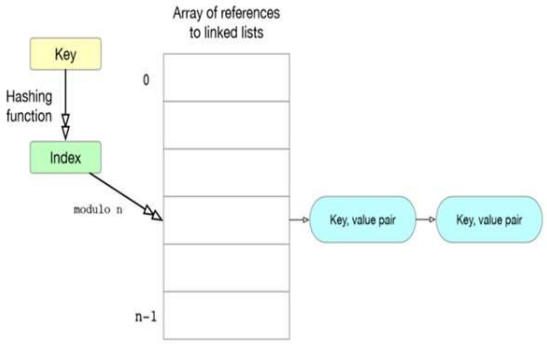
- Just like a list you can iterate over the keys of the dictionary (more cases are given in Flow control Structures):

In [ ]:
```
d = { 1: "Hello", 2: "world" }
d
for a1 in d.keys():
    print(a1)
```

University of Reading

Dictionaries as Hash Tables

- Dictionaries are implemented as hash tables:
  - Keys are mapped to integers using a hashing function.
  - This value is then mapped into an array using a modulo operation.
  - Occasionally different keys map to the same value.
  - When keys maps to the same value, use a linked list of (key, value) pairs.



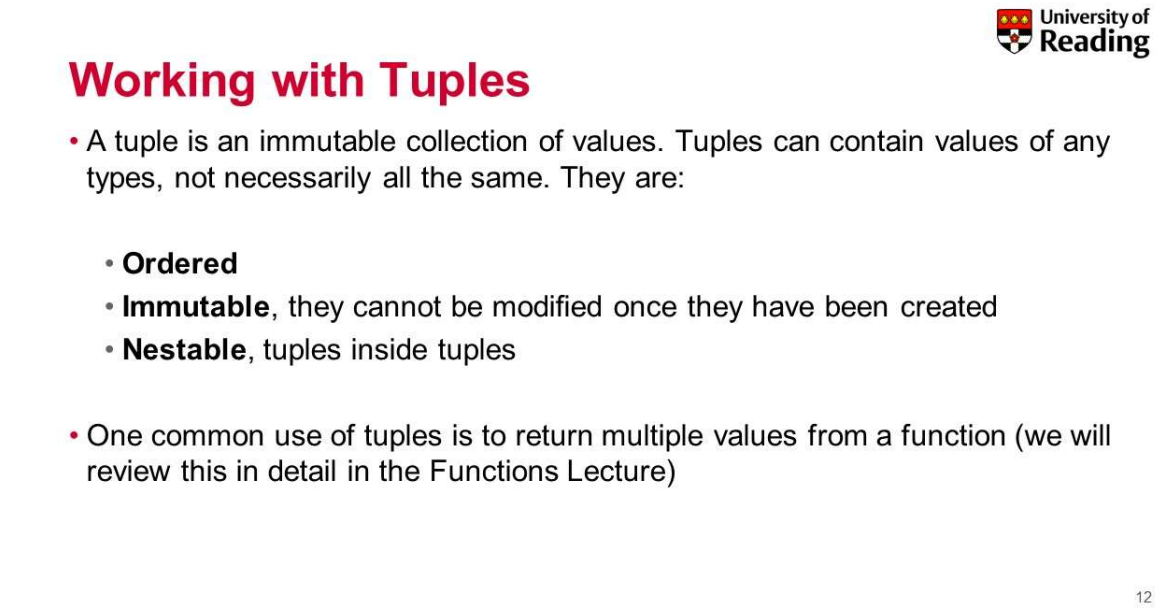
10

University of Reading

Working with Dictionaries

- As a result of the hash table implementation dictionaries in Python are fast:
  - I Lookups take on average time O(1)
  - Assignments take on average time O(1)
- However in the worst case, where all keys collide, these operations take time O(n).

11



Tuples initialisation

- Tuples are created and indexed just like lists:

In [ ]:
```
a = (1,"Snake",[4,5,6])
a
```

- You can use the unpacking syntax for lists on tuples too:

In [ ]:
```
b, c, d = a
print(a)
print(b)
print(c)
print(d)
```

Lookup:

- Tupples can be accessed same as lists

In [ ]:
```
a = (1,"Snake",[4,5,6])
a
```

In [ ]:
```
a[1]
```

In [ ]:
```
a[0]
```

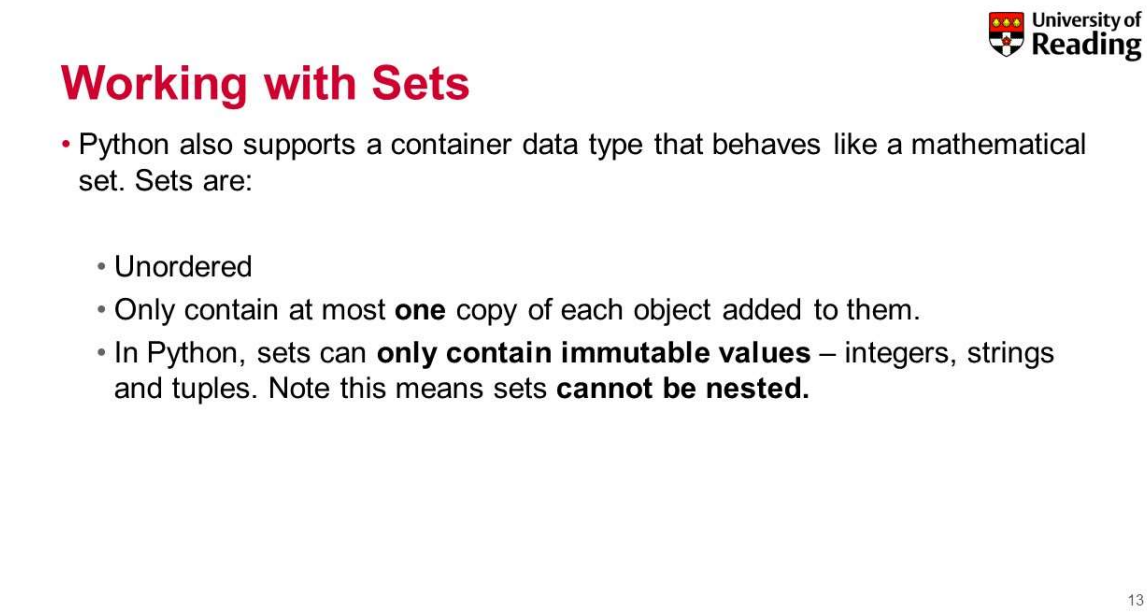
In [ ]:
```
a[0:2]
```

In [ ]:
```
a[2][2]
```

Updating

- Tuples are **not mutable** so they can not be modified once they are created.

In [ ]:
```
a[0] = 5
```



Sets initialisation

- Sets can be created using an initialisation similar to dictionaries:

In [ ]:
```
a = {1,3,2,"Hello"}
a
```

- You can also use the set constructor to build sets, and create empty sets:

In [ ]:
```
b = set() # An empty set
c = set([1,3,2,"Hello"])
#b
c
```

In [ ]:
```
##Only one copy of each object added to them
d = {1,1,2,3,4}
d
```

University of Reading

Working with Sets

- Sets support mathematical operations on sets like:
  - Union
  - Intersection
  - Difference
  - Membership test
- We can also use the add() method to add objects to a set. All these operations preserve the condition that objects can only appear once in a set.

14

In [ ]:
```
a = {1,3,7,11}
b = {3,5,7,9}
```

In [ ]:
```
##Union:
a | b
```

In [ ]:
```
##Intersect
a & b
```

In [ ]:
```
#Difference
a - b
b - a
```

In [ ]:
```

```

In [ ]:
```
##Membership test
7 in a
```

In [ ]:
```
##Add
b.add(7)
b
```

- One useful application of sets in Python is to filter lists to a set containing only the unique items in the list:

In [ ]:
```
a = [1,3,1,8,5,3,5]
b = set(a)
b
#a
```

University of Reading

Logical and Comparison Operators

- Logical:
  - AND
  - OR

- Comparison Operators
  - Equal
  - Not Equal
  - Less than
  - Greater than

15

In [ ]:
```
(1 > 2) and (2 <= 3)
```

In [ ]:
```
(1 >= 2) & (2 < 3)
```

In [ ]:
```
(1 == 2) or (2 == 3)
```

In [ ]:
```
(1 != 2) | (2 == 3)
```

In [ ]:
```
not (1 != 2) | (2 == 3)
```

University of Reading

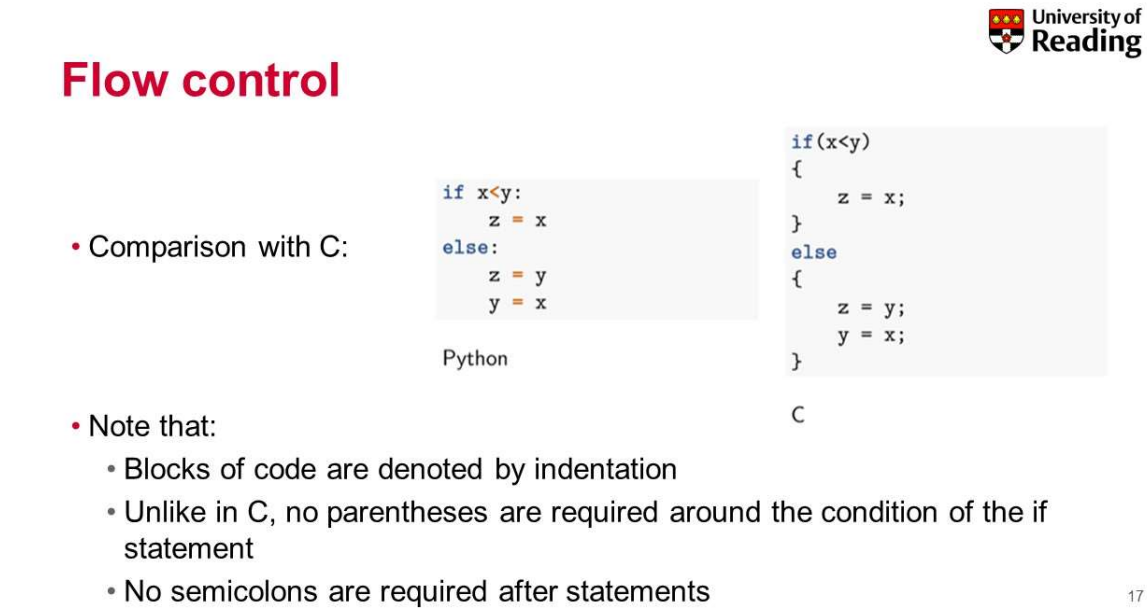
Flow control

- As well as assignment statements seen above, Python supports most of the flow control structures used in C and Java like:

  - If-Else statements
  - For loops
  - While loops

- **Built-in statements in Python like if and for use a syntax that has a header line followed by a :, then an indented block of text.**

16



If example:

Simple IF:

In [ ]:
```
##GRADE System:
grade = 38
if grade >= 40:
    print('Pass')
else:
    print('Fail')
```

Nested if:

In [ ]:
```
##GRADE System:
grade = 50
if grade >= 40:
    if grade > 49:
        print('Second Class Honours or more')
    else:
        print('Third Class Honours')
else:
    print('Fail')
```

Elif

In [ ]:
```
##GRADE System:
grade = 52
if grade < 40:
    print('Fail')
elif grade <= 49:
    print('Third Class Honours')
else:
    print('Second Class Honours or more')
```

For Loop

- For loops in Python can be applied over a range
  - range(N) generates an iterator starting at 0 and ending at N−1I
  - **range(N) doesn't include N!**

In [ ]:
```
range(10)
```

In [ ]:
```
for i in range(8):
    print(i)
```

- For loops can also be applied over other objects that can be iterated over:
  - For a list xs, we can iterate over each member of xs in order

In [ ]:
```
##For loops in lists - values:
xs = [1,4,2,3]
#xs
for x in xs:
    print(x)
```

In [ ]:
```

```

In [ ]:
```
##For loops in lists - index and values:
xs = [1,4,2,3]
for a1, a2 in enumerate(xs):
    print(a1, a2)
```

In [ ]:
```
##For loops in set
a = {1,3,2,"Hello"}
for a1 in a:
    print(a1)
```

In [ ]:
```
##For loops in Tuples
a = (1,"Snake",[4,5,6])
for a1 in a:
    print(a1)
```

In [ ]:
```
##For loops in Dictionaries-Keys:
d = { 1: "Hello", 2: "world" }
for a1 in d:
    print(a1)
```

In [ ]:
```
##For loops in Dictionaries-Keys and Values:
d = { 1: "Hello", 2: "world" }
for a1, a2 in d.items():
    print(a1, a2)
```

In [ ]:
```
d.items()
```

While loop

- While loops execute a block of code as long as a condition is true

In [ ]:
```
counter = 5
while counter > 0:
    print("Hello world")
    counter = counter - 1
```

Break

Just as in C and Java you can break a for or while loop to exit the code block immediately:

In [ ]:
```
counter = 5
while counter > 0:
    print("Hello world", counter)
    counter = counter - 1
    if counter < 3:
        break
    print('Still alive')
print('Loop ended, final value of counter:', counter)
```

Continue

- The continue statement allows you to continue from the top of a for or while block:

In [230…
```
counter = 5
while counter > 0:
    print("Hello world", counter)
    counter = counter - 1
    if counter < 3:
        continue
    print('Still alive')
print('Loop ended, final value of counter:', counter)
```

```
Hello world 5
Still alive
Hello world 4
Still alive
Hello world 3
Hello world 2
Hello world 1
Loop ended, final value of counter: 0
```

Loop else

- A nice space saving feature in Python is the else statement for loops.
- The else after a for loop is **only reached if the loop ends without a break statement** being encountered.

In [237…
```
ls = ["t","g","e","o","i","p"]

for l in ls:
    if l=="t":
        found = True
        break
else:
    found = False
```

```
found
```

Out[237… True

University of Reading

Summary

- Python offers a broad datatype options that can be implemented depending on the requirement of the task to perform.
- The main difference between the data types are their characteristics such as mutable, ordered and nested.
- Different examples were cover to identify the differences between flow control structures and logical and comparison operators and the way this tools can be implemented.

18

University of Reading

Questions

19

University of Reading

References

- Thomas McKinney, Wes 'Python for data analysis : data wrangling with Pandas, NumPy, and IPython'.
- Learn by example (2019). Python List Slicing. Learn by Example. Online at: https://www.learnbyexample.org/wp-content/uploads/python/Python-List-Slicing-Illustration.png. Accessed 03/08/2020.

In [ ]: