COMP10020
Introduction to Programming II
Python Data & Data Structures

Dr. Brian Mac Namee
brian.macnamee@ucd.ie
School of Computer Science
University College Dublin

BASIC PYTHON
VARIABLES AND TYPES

Variables

Variables allow us to store values

```
a = 17
b = 25
c = a + b
```

As soon as a variable is used in an assignment it is created

Variable name rules

- Composed of letters, numbers, and underscore characters
- Start with either a letter or an underscore
- Case sensitive
- Not a Python keyword (e.g. **if**, **while**, **lamda** …)

Quiz

Which variable names are allowed?

a) my_var
b) 7eleven
c) __
d) total income
e) fileName
f) file_name
g) fIlEnAmE
h) my.favourite

Variables

Everything in Python is an object

- A variable is a reference to an object
- We can use the **id** function to see the address of an object
- Objects have types, variables do not
- We can use the **type** function to check the type of the object a variable references

Types

Python supports the following basic types

- None
- Integers
- Floating-point numbers
- Complex numbers
- Booleans
- Strings

Boolean Operators

Using Boolean objects opens up Boolean operators

Operator	Description
x == y	equals
x != y	not equals
not x	not
x or y	or
x and y	and
x ^ y	exclusive or

Python Strings

In Python strings are stored as sequences of characters

- Sequences are 0 indexed

Python Strings

In Python strings are stored as sequences of characters

- Sequences are 0 indexed

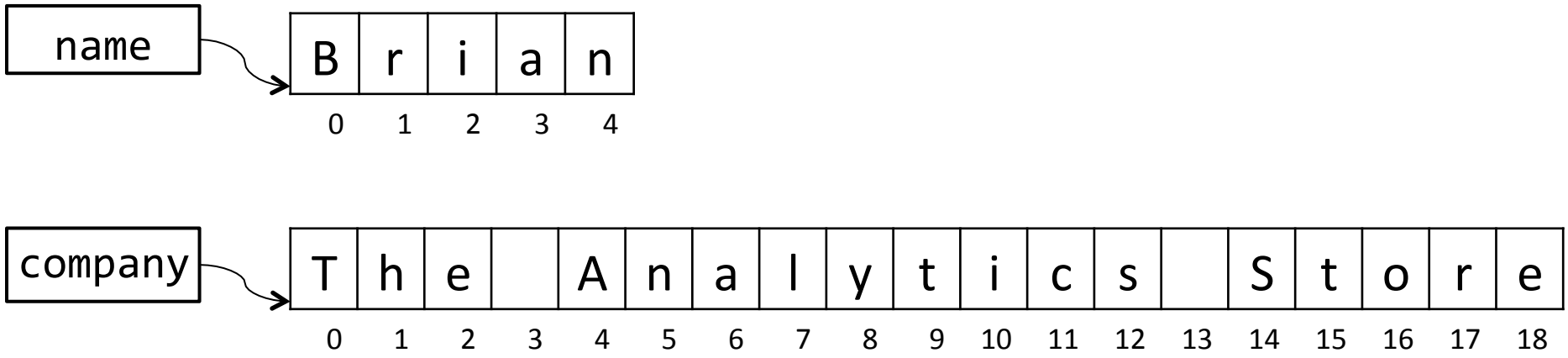
```
name = "Brian"
company = 'The Analytics Store'
```

Python Strings

In Python strings are stored as sequences of characters

- Sequences are 0 indexed

```
name = "Brian"
company = 'The Analytics Store'
```



Python Strings

Key things to note about strings

- Individual characters are accessed using square brackets

```
name[0]
```

- Range slice notation can be used to extract substrings

```
name[1:4]
```

- Strings are immutable
- The length of a string can be accessed using the **len** function

Python String Operators

There are some neat string operators:

- **+** performs string concatenation
- ***** repeats strings
- **in** checks if string contains another
- **not in** checks if string does not contain another

Useful String Functions

Python provides multiple string manipulation functions

- **count**(str, beg= 0 , end = len(string))
  Count the occurrences of str in the string
- **find**(str, beg = 0, end = len(string))
  Searches for occurrence of str in the string and returns its index (or -1 if not found)
- **startswith / endswith**(str, beg = 0, end = len(string))
  Checks if a string starts with str
- **strip**()
  Removes whitespace from the beginning and end of a string

Useful String Functions

- **split**(delim)
  Splits the string using delim
- **upper**() / **lower**() / **title**()
  Converts the string to upper/lower/title case
- **isalnum**() / **isdigit**() / **isalpha**()
  Tests whether or not the string contains alphanumeric/numeric/alpha characters only
- **min**() / **max**()
  Returns min/max alphabetical character
- **replace**(old, new)
  Replaces all occurrences of characters old with new
- **translate**(table)
  Translates the string according to a translation table - use **maketrans** to make a table

LISTS

Lists

A *list* is an ordered collection of other variables.

These variables can have different types.

Lists definitions are enclosed within square brackets, [ ]

Lists

mylist = []

Lists

numbers = [12, 108, 21]

0	12
1	108
2	21

Lists

somedata = ["text", 7, 0.34, True]

0	“text”
1	7
2	0.34
3	True

Lists

values = [34, 9, 12, 34]

0	34
1	9
2	12
3	34

Lists

fulllist = [9, 12, 23, 18, 21]

0	9
1	12
2	23
3	18
4	21

Lists

values = [34, 9, 12, 34]

0	34
1	9
2	12
3	34

Lists

values[2] = 5000

0	34
1	9
2	5000
3	34

Lists

values.append("extra")

0	34
1	9
2	5000
3	34
4	“extra”

Lists

values + [11, 27]

0	34
1	9
2	5000
3	34
4	“extra”

0	11
1	27

Lists

values + [11, 27]

0	34
1	9
2	5000
3	34
4	“extra”

0	11
1	27

0	34
1	9
2	5000
3	34
4	“extra”
5	11
6	27

Lists

mylist = [3,6,9,12]

0	3
1	6
2	9
3	12

Lists

letters = ["b","d","a","c"]

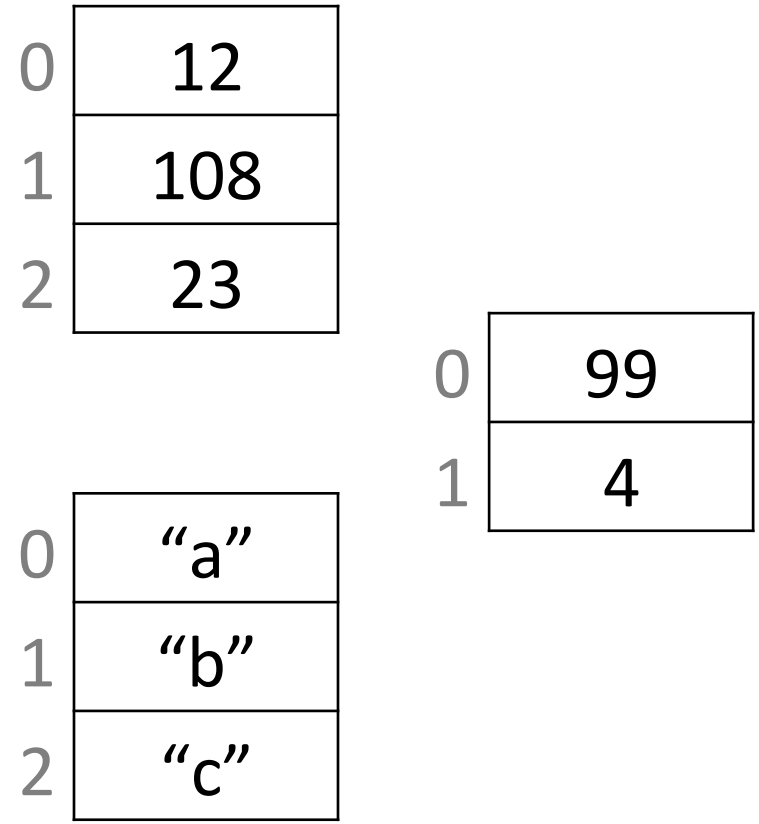
0	“b”
1	“d”
2	“a”
3	“c”

Lists

child1 = [12, 108, 23]

child2 = [99, 4]

child3 = ["a","b","c”]



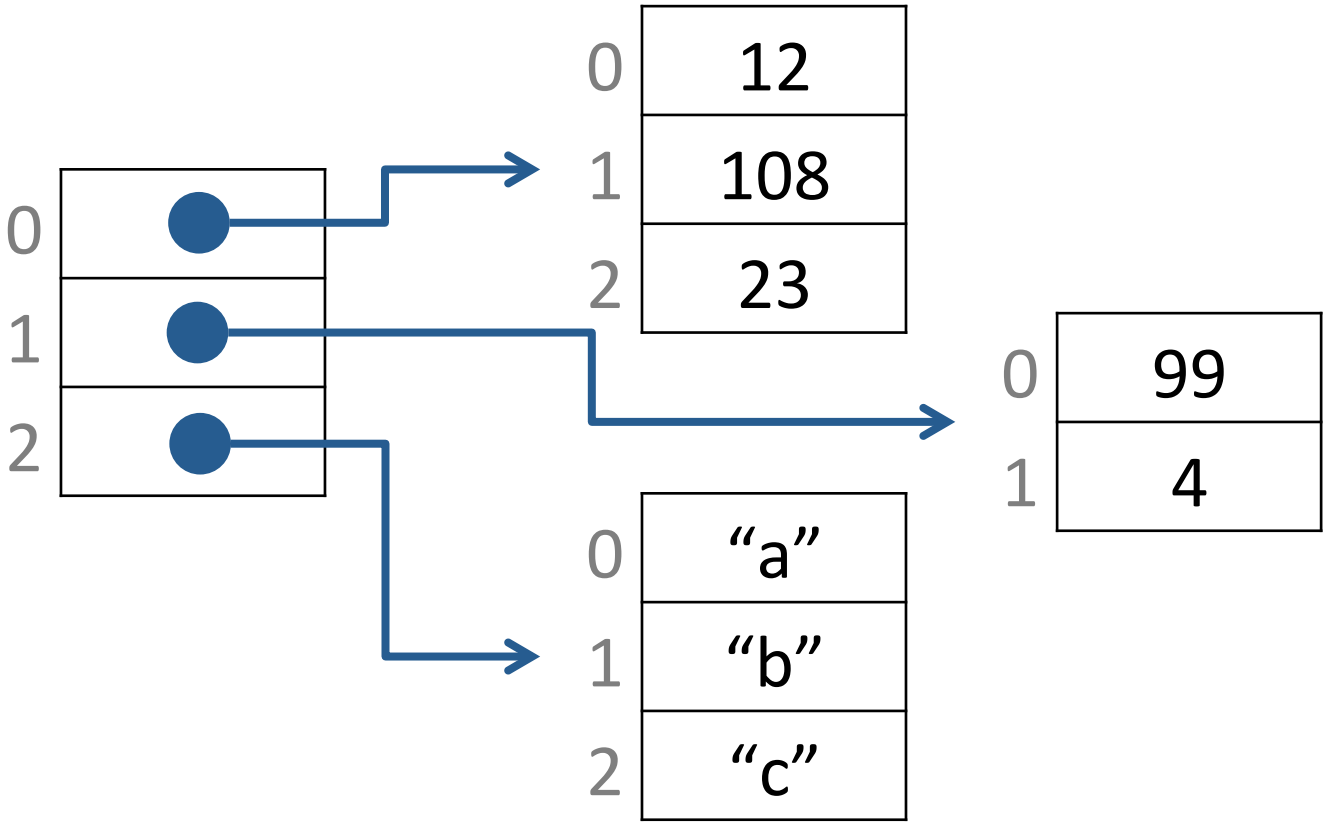
Lists

child1 = [12, 108, 23]

child2 = [99, 4]

child3 = ["a","b","c”]

parent = [child1, child2, child3]



TUPLES

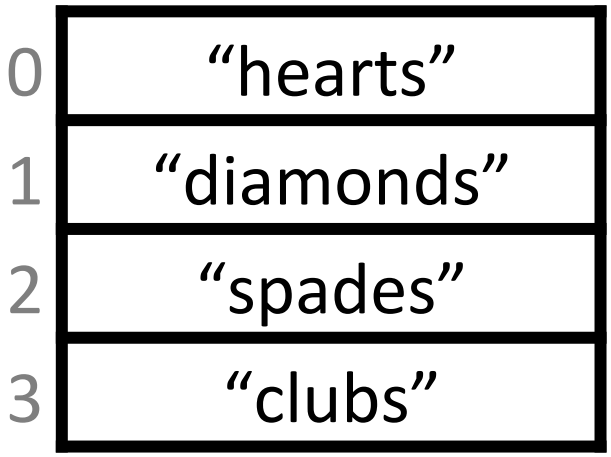
Tuples

Tuples are like lists but are "immutable" - this means that once they are created, they cannot be modified

Tuples are created using parenthesis notation, ()

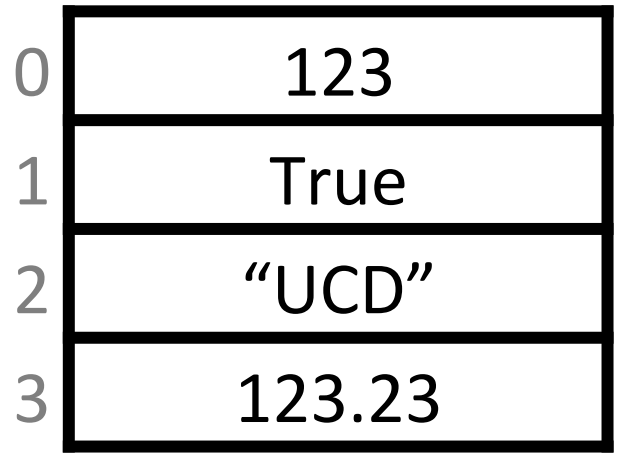
Tuples

suits = ("hearts", "diamonds", "spades", "clubs”)



Tuples

t = (123, True, "UCD", 123.23)



SETS

Sets

Sets are unordered lists which contain no duplicate values

They can be created from lists, strings or any other iterable value, using the *set* function

Sets do not have an order, so we cannot index into them by position

Sets

mylist = [1,3,1,4,3,6,8,1,4,4]

0	1
1	3
2	1
3	4
4	5
5	6
6	8
7	1
8	4
9	4

Sets

set(mylist)

0	1
1	3
2	4
3	6
4	8

Sets

set("abcddabcdaacbcc")

0	“a”
1	“b”
2	“c”
3	“d”

Sets

names = set(['Bill','Lisa','Ted'])

0	“Bill”
1	“Lisa”
2	“Ted”

Sets

x = set([1,2,3,4])

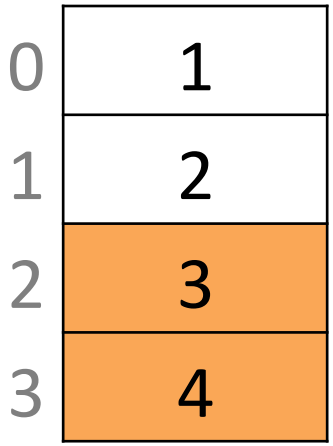
0	1
1	2
2	3
3	4

y = set([3,4,5])

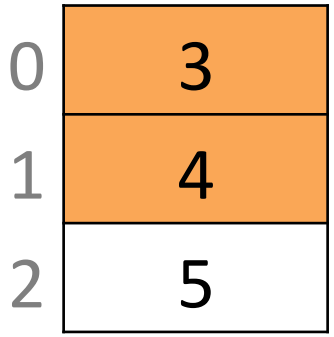
0	3
1	4
2	5

Sets

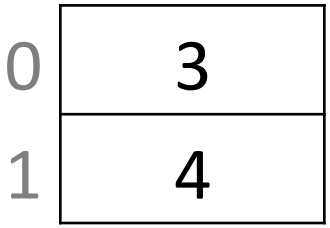
x = set([1,2,3,4])



y = set([3,4,5])

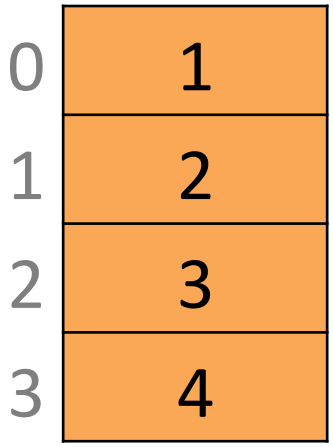


x.intersection(y)

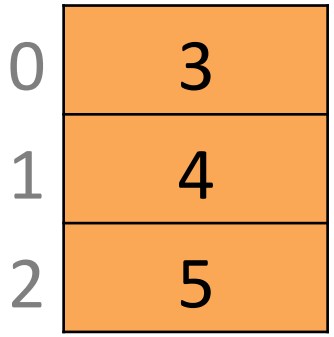


Sets

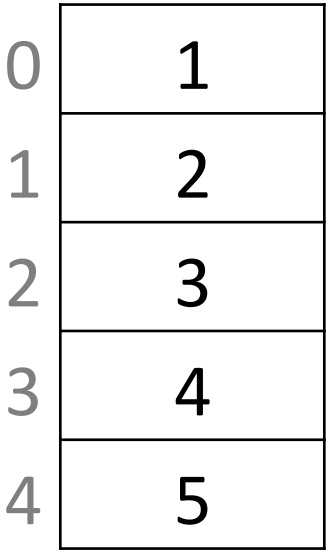
x = set([1,2,3,4])



y = set([3,4,5])



x.union(y)



Sets

x = set([1,2,3,4])

0	1
1	2
2	3
3	4

y = set([3,4,5])

0	3
1	4
2	5

x.difference(y)

0	1
1	2

Sets

x = set([1,2,3,4])

0	1
1	2
2	3
3	4

y = set([3,4,5])

0	3
1	4
2	5

y.difference(x)

0	5

DICTIONARIES

Dictionaries

A *dictionary* (sometimes called a *map*) is a data structure containing an unordered set of *(key,value)* pairs

Each *key* is linked to a *value*

The keys and values can be any basic Python variable.

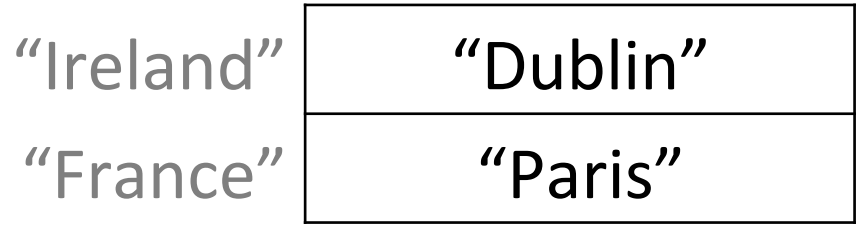
Dictionaries can be created using curly bracket notation { }, and can either be initially empty or populated with one or more pairs

Dictionaries

d0 = {}

Dictionaries

```
d1 = {"Ireland":"Dublin", "France":"Paris"}
```



Dictionaries

d2 = {"age": 22, "name": "alice", “employed”: False}

“age”	22
“name”	“alice”
“employed”	False

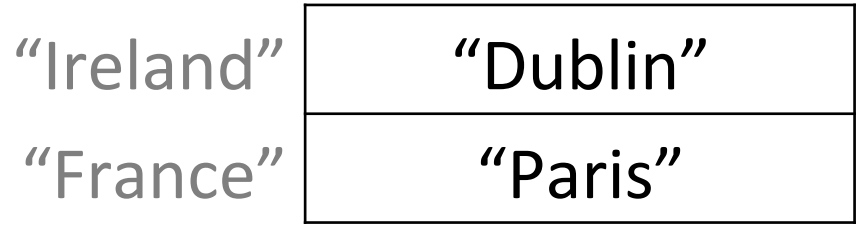
Dictionaries

mixedmap = {1:"ucd", 0.8:False, "b":10, "c":"d”}

1	“ucd”
0.8	False
“b”	10
“c”	“d”

Dictionaries

```
d1 = {"Ireland":"Dublin", "France":"Paris"}
```



Dictionaries

d1["Germany"] = "Berlin"

“Ireland”	“Dublin”
“Germany”	“Berlin”
“France”	“Paris”

Dictionaries

d1[”Ireland"] = ”Galway"

“Ireland”	“Galway”
“Germany”	“Berlin”
“France”	“Paris”

SUMMARY

Summary

Python offers a range of different basic data structures

Choosing the right data structure in which to store different types of data is always important