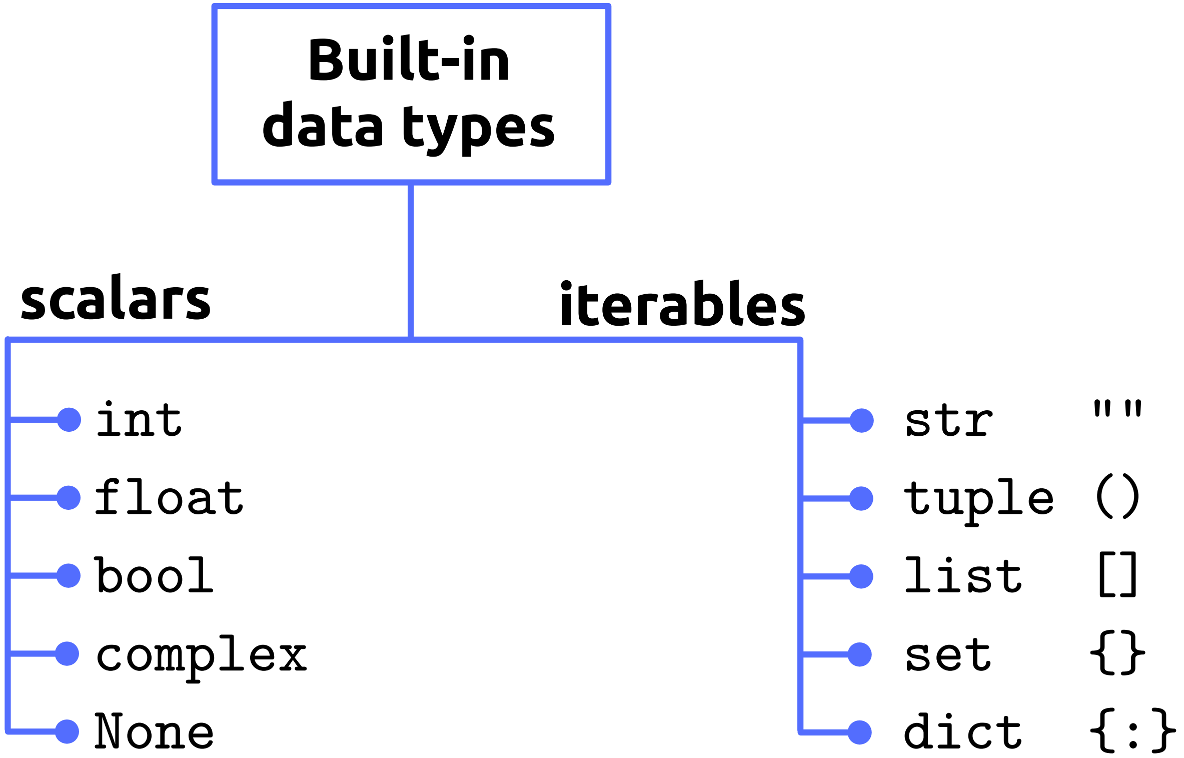
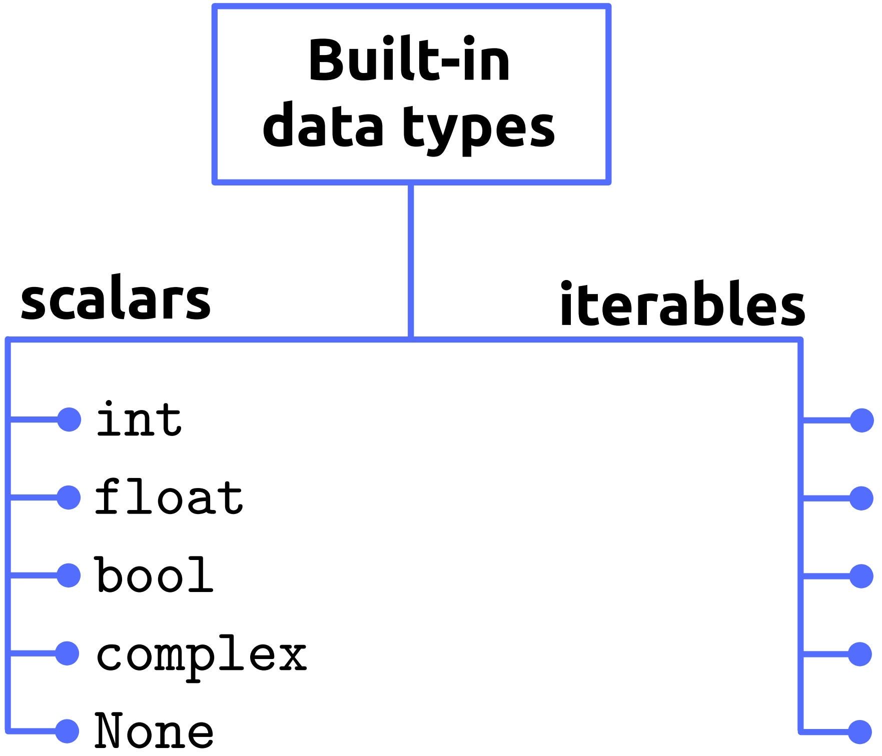
235A Python for Engineers

Module 2:
Python basics


Built-in data types
scalars
iterables
int
float
bool
complex
None
str ""
tuple ()
list []
set {}
dict {:}

Mutability and Order of iterables



		mutable	sequence
str	""	✗	✓
tuple	()	✗	✓
list	[]	✓	✓
set	{}	✓	✗
dict	{:}	✓	✗

Checking membership in an iterable

Syntax:

```
<value> in <iterable>
```

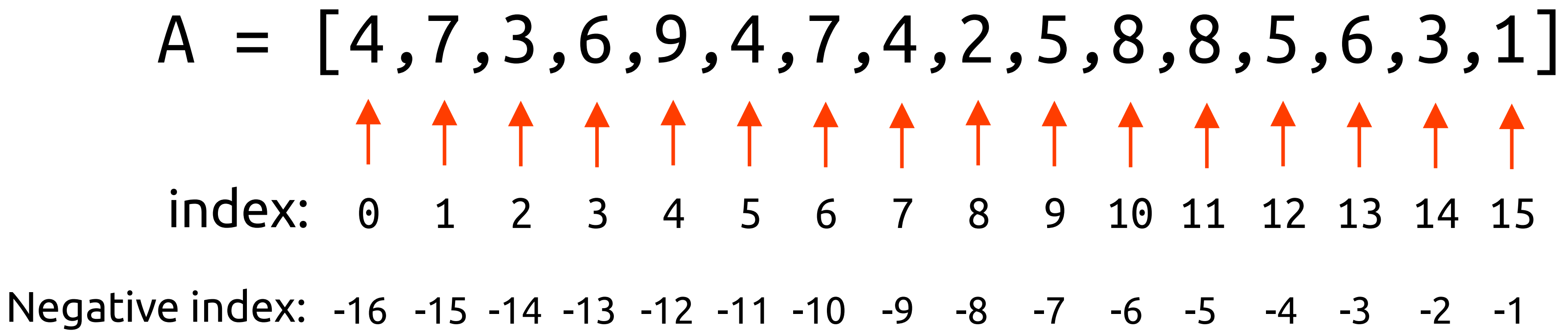
Indexing iterables

Description: Get the i’th element in an ordered iterable `A`

Syntax: `A[index]`

Note: Indexing is 0-based

Applies to: Sequences



```
A = [4,7,3,6,9,4,7,4,2,5,8,8,5,6,3,1]
```

index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Negative index:	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Slice indexing

Description: Extract a sub-iterable from an ordered iterable `A`.

Syntax:

```
A[start=0:stop=len(A)]
```

```
A[start=0:stop=len(A):step=1]
```

Note: ‘stop’ value is not included in the result

Examples

```
A = [4,7,3,6,9,4,7,4,2,5,8,8,5,6,3,1]
```

index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Negative index:	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Finding an element in an ordered iterable

Description: Return the index of the first instance of `<value>` in A.

Syntax:

```
A.index(<value>)
```

Note: Throws an error if `<value>` not in A

List methods

• `[]` or `list()`	… Make an empty list.
• `A.`**`append`**`(value)`	… Puts value at the end of `A`.
• `A.`**`extend`**`(iterable)`	… Appends each value of iterable to `A`.
• `A.`**`insert`**`(index,value)`	… Inserts value at a `A[i]`.
• `A.`**`remove`**`(value)`	… Remove the first instance of value from `A`.
• `A.`**`pop`**`(index)`	… Extract the item at index and return it.
• `A.`**`clear`**`()`	… Remove all items from `A`.

When to use a tuple instead of a list?

- Tuples are immutable, lists are mutable.
- Tuples are smaller and faster than lists.
- Use tuples as keys to dictionaries

Unpacking ordered iterables

Description: Shorthand syntax for assigning the elements of an ordered iterable to respective variables.

Syntax:

```
X1,…,Xn = A
```

Note: Will fail if `len(A)!=n`

Dictionaries

Description: Set of key-value pairs.

Syntax:

```
A = {key1:value1, key2:value2, …, keyN:valueN}
```

Note: Keys must be unique.

Joining and splitting strings

String + **Description**: Join two or more strings.

Syntax:

```
A = str1 + str2 + … + strN
```

Returns: A string.

split() **Description**: Split a string at a delimiter.

Syntax:

```
A.split(<delimiter>)
```

Returns: A list of strings.

Formatting strings

Description: Build strings with numerical values of variables

Syntax:
```
A = {key1:value1, key2:value2, …, keyN:valueN}
```

Note: Keys must be unique.

On the use of whitespace in Python

- Most languages use special symbols to demarcate blocks of code.
  - C, C++, Java: `{}`
  - Matlab: `end`
- Python uses **indentation levels**.
- Common practice: a **tab character** or **4 white spaces.**
- **Consistency** is important.

“if” statements

An **“if”** statement (or “**conditional”** statement) selects one of several blocks of code to execute, according to respective boolean expressions.

Syntax:

```
if <boolean expression 1>:
    <code block 1>
elif <boolean expression 2>:
    <code block 2>
    …
elif <boolean expression N-1>:
    <code block N-1>
else:
    <code block N>
```

“while” loops

A **“while”** loop executes a block of code as long as a boolean expression evaluates to `True`.

Syntax:

```
while <boolean expression>:
    <code block>
```

“for” loops

A **“for**” loop executes the block of code as many times as there are items in a given iterable. A variable is assigned successive values from the iterable.

Syntax:

```
for <variable> in <iterable>:
    <code block>
```

Note: Order of execution is only guaranteed for sequences.

“break” and “continue”

Used within loops (both **for** and **while**).

- **break**: exit the **for** (or **while**) loop immediately.
- **continue**: ignore the rest of the block and go on to next iteration.

“range”

Description: Generate a uniformly spaced list of numbers.

Syntax:
```
range(stop)
```
```
range(start,stop,step=1)
```

“enumerate”

Description: Iterate simultaneously through the index and values of a sequence.

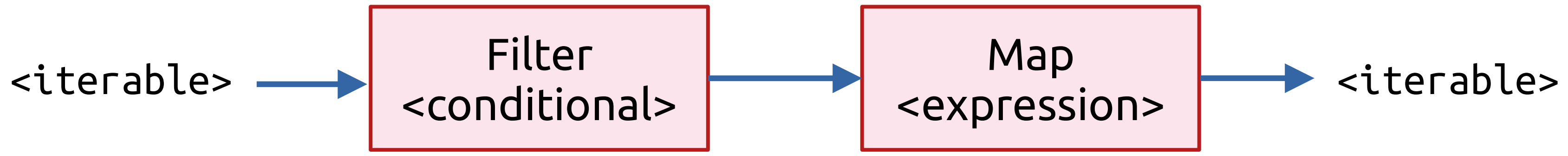
Syntax:
```
for index, value in enumerate(A):
    <code block>
```

Comprehensions

Description: A succinct syntax for creating iterables from other iterables.

Syntax:

set:	`{<expression> for <var> in <iterable> if <conditional>}`
tuple:	`(<expression> for <var> in <iterable> if <conditional>)`
list:	`[<expression> for <var> in <iterable> if <conditional>]`
dict:	`{key:value for <var> in <iterable> if <conditional>}`


<iterable>
Filter
<conditional>
Map
<expression>
<iterable>