COMP1730/COMP6730
Programming for Scientists

Strings and more on sequences

Announcements

* Homework 3 is due on Sunday (27th Aug, 11:55pm).
  - **Good code quality is required to get full marks!**
* From now on, some students may be randomly selected for in-lab **oral assessment of homework**.
* Lab this week will be large, working time outside 2 hours lab is expected.
  - **Make use of CodeBench!** Tutor may discuss your submitted CodeBench the following week.

Lecture outline

* Character encoding & strings
* Indexing, slicing & sequence operations
* Iteration over sequences

Characters & strings

Strings

- Strings – values of type `str` in python – are used to store and process text.
- A string is a *sequence* of *characters*.
  - `str` is a sequence type.
- String literals can be written with
  - single-quotes, as in `'hello world'`
  - double-quotes, as in `"hello world"`
  - triple quotes for multi-line strings: `'''hello world'''` or `"""hello world"""`
  - Beware of copy–pasting code from slides (and other PDF files or web pages).

- Quoting characters other than those enclosing a string can be used inside it:

```
>>> "it's true!"
>>> '"To be," said he, ...'
```

- Quoting characters of the same kind can be used inside a string if escaped by backslash (\):

```
>>> 'it\'s true'
>>> "it's a \"quote\""
```

- Escapes are used also for some non-printing characters:

```
>>> print("\t1m\t38s\n\t12m\t9s")
```

Australian National University

Unicode, encoding and font

- *Unicode* defines numbers ("*code points*") for >140,000 characters (in a space for >1 million).

Byte(s)	Code point	Glyph
0100 0101	69	EEE$\mathcal{E}$
1110 0010 1000 0010 1010 1100	8364	€€€€

- python 3 uses the unicode character representation for all strings (a major change from python 2).
- Functions `ord` and `chr` map between the character and integer representation:

```
>>> ord('A')
>>> chr(65 + 4)
>>> chr(32)
>>> chr(8364)
>>> chr(20986)+chr(21475)
>>> ord('3')
```

- See `unicode.org/charts/`.

More about sequences

Three built-in sequence type in Python:

* `list`: ['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd']
* `tuple`: ('H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd')
* `str`: "Hello World"
* `str` and `tuple` are **immutable**.

Indexing & length (reminder)

characters	H	e	l	l	o		W	o	r	l	d
index	0	1	2	3	4	5	6	7	8	9	10
									...	−2	−1

FIGURE 4.1 The index values for the string `'Hello World'`.

Image from Punch & Enbody

* In python, all sequences are indexed from 0.
* ...or from end, starting with -1.
* The index must be an integer.
* The length of a sequence is the number of elements, *not* the index of the last element.

Slicing

* Slicing returns a subsequence:

  `s[`*`start`*`:`*`end`*`]`

  - *`start`* is the index of the first element in the subsequence.
  - *`end`* is the index of the first element after the end of the subsequence.
* Slicing works on all built-in sequence types (`list`, `str`, `tuple`) and returns the same type.
* If *`start`* or *`end`* are left out, they default to the beginning and end (i.e., after the last element).

* The slice range is “half-open”: start index is included, end index is one after last included element.

```
>>> s = "Hello World"
>>> s[6:10]
'Worl'
```

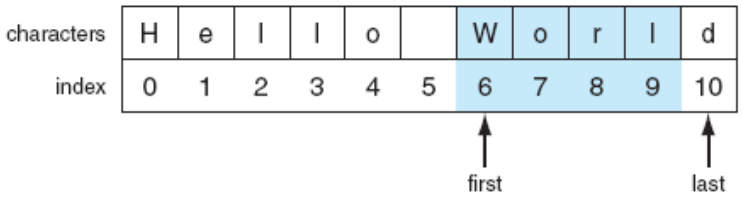

FIGURE 4.2 Indexing subsequences with slicing.

Image from Punch & Enbody

Australian National University

* The end index defaults to the end of the sequence.

```
>>> s = "Hello World"
>>> s[6:]
'World'
```

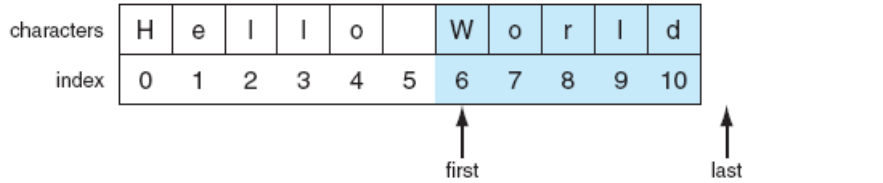

Image from Punch & Enbody

Australian National University

* The start index defaults to the beginning of the sequence.

```
>>> s = "Hello World"
>>> s[:5]
'Hello'
```

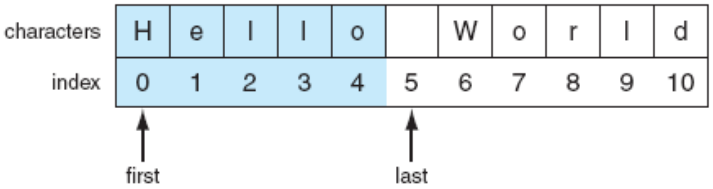

Image from Punch & Enbody

Australian National University

```
>>> s = "Hello World"
>>> s[9:1]
''
>>> s[-100:5]
'Hello'
```

* An empty slice (index range) returns an empty sequence
* Slice indices can go past the start/end of the sequence without raising an error.

```
>>> s = "Hello World"
>>> s[3:-3]
'lo Wo'
>>> s[-8:-3]
'lo Wo'
>>> s[-8:-8]
''
```

Slicing also works with three arguments:

```
s[start:end:stepBy]
```

```
>>> s = "Hello World"
>>> s[0:11:2]
```

Australian National University

Operations on sequences

* Reminder: *value types determine the meaning of operators applied to them*.
* Concatenation: `seq + seq`

```
>>> "comp" + "1730"
>>> [1,2] + [3,4,5]
```

* Repetition: `seq * int`

```
>>> "Oi! " * 3
>>> (1,3) * 2
```

* Membership: `value in seq`
  - *Note:* `str in str` tests for substring.
* Equality: `seq == seq`, `seq != seq`.
  - Two sequences are equal if they have the same length and equal elements in every position.

Australian National University

Sequence comparisons

* Comparison (same type): `seq < seq`, `seq <= seq`, `seq > seq`, `seq >= seq`, returning `True` or `False`.
* `seq1 < seq2` if
  - `seq1[i] < seq2[i]` for some index `i` and the elements in each position before `i` are equal; or
  - `seq1` is a prefix of `seq2`.
* : Question: What is the value of:

```
>>> [1,2] < [1, 3]
```

Australian National University

String comparisons

* Each character corresponds to an integer.

```
ord(' ') == 32
ord('A') == 65
ord('Z') == 90
ord('a') == 97
ord('z') == 122
```

* Character comparisons are based on this.

```
>>> "the ANU" < "The anu"
>>> "the ANU" < "the anu"
>>> "nontrivial" < "non trivial"
```

The `enumerate` function

* The `enumerate` function takes a sequence and returns a representation of a sequence of `(index, element)` pairs.
  - Use `for` with multiple assignment.

```
for index,char in enumerate("The quick brown fox"):
    print("at", index, "we have", char)

# instead of
s = "The quick brown fox"
for index in range(len(s)):
    print("at", index, "we have", s[index])
```

Australian National University

String methods

* Methods are only functions with a slightly different call syntax:

```
"Hello World".find("o")  # return lowest index of occurrence
"Hello World".count("o") # count non-overlapping occurrences
```

instead of

```
str.find("Hello World", "o")
str.count("Hello World", "o")
```

* python's built-in types, like `str`, have many useful methods.
  - `help(str.find)`
  - `help(str.count)`
  - `docs.python.org`

Australian National University

Programming problem (bioinformatics):

* a *k-mer* is a substring of length *k* of a longer DNA sequence (`https://en.wikipedia.org/wiki/K-mer`).
* For a DNA sequence, find all distinct k-mers and their number of occurrences.

```
AGAGACCCCCT
AGA
 GAG
  AGA
   GAC
    ACC
     CCC
      CCC
       CCC
        CCT
```

→

k=3:

```
AGA 2
GAG 1
GAC 1
ACC 1
CCC 3
CCT 1
```

k=1:

```
A 3
G 2
C 5
T 1
```