



Australian  
National  
University

# COMP1730/COMP6730

## Programming for Scientists

# Strings



# Announcements

- \* Homework 3 is due next Monday (26th Aug, 11:55pm).
- \* Week 6&7 conflict questionnaire is due this Friday (23rd Aug). **But do not fill out if you have no conflict.**
- \* Lab this week will be large, working time outside 2 hours lab is required.
- \* Drop-in consultation: every Monday, 4pm, room 1.17 Hanna Neumann blg.



# Lecture outline

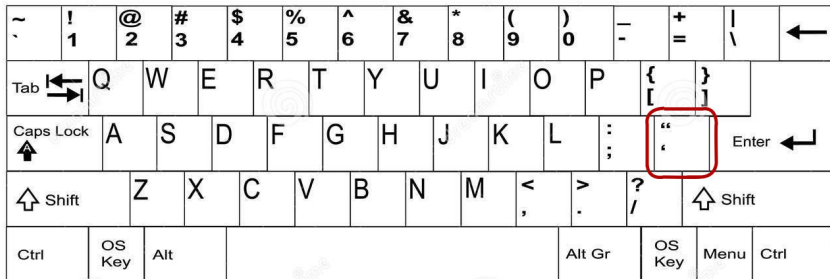
- \* Character encoding & strings
- \* Indexing, slicing & sequence operations
- \* Iteration over sequences



# 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 there'`
  - double quotes, as in `"hello there"`
  - triple quotes, as in `'''hello there'''`



- \* 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")
```

# Character encoding

- \* Idea: Every character has a number.
- \* Baudot code (1870).
- \* 5-bit code, but also sequential (“letter” and “figure” mode).

V	IV		I	II	III
		A	1	●	
		É	&	● ●	
		E	2		●
		I	<u>0</u>		● ●
		O	5	● ● ●	
		U	4	●	●
		Y	3		●
●		B	8		●
●		C	9	●	●
●		D	0	● ● ●	
●		F	<u>f</u>		● ●
●		G	7		●
●		H	<u>h</u>	● ●	
●		J	6	●	
●		Figure	Blank		

● ●	Erasure	Erasure		
● ●	K	(	●	
● ●	L	=	● ●	
● ●	M	)		●
● ●	N	N°		● ●
● ●	P	%	● ● ●	
● ●	Q	/	●	●
● ●	R	-		●
●	S	:		●
●	T	!	●	●
●	V	'	● ● ●	
●	W	?		● ●
●	X	,		●
●	Z	:	● ●	
●	<u>t</u>	.	●	
●	Blank	Letter		



# Unicode, encoding and font

- \* *Unicode* defines numbers (“*code points*”) for >120,000 characters (in a space for >1 million).

Byte(s)	Code point	Glyph
0100 0101	69	EEEE€
1110 0010		
1000 0010		
1010 1100	8364	€€€€

- \* python 3 uses the unicode character representation for all strings.
- \* 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/](http://unicode.org/charts/).



# More about sequences

# 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.

- \* `len(sequence)` returns sequence length.
- \* Sequence elements are accessed by placing the index in square brackets, `[]`.

```
>>> s = "Hello World"
```

```
>>> s[1]
```

```
'e'
```

```
>>> s[-1]
```

```
'd'
```

```
>>> len(s)
```

```
11
```

```
>>> s[11]
```

```
IndexError: string index out of range
```

# 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 end index defaults to the end of the sequence.

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

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

↑  
first

↑  
last

Image from Punch & Enbody







```
>>> 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.

# Operations on sequences

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

```
>>> "comp" + "1730"
```
- \* Repetition:  $seq * int$   

```
>>> "Oi! " * 3
```
- \* Membership:  $value \text{ in } seq$   
– Note:  $str \text{ in } str$  tests for substring.
- \* Equality:  $seq == seq$ ,  $seq != seq$ .
- \* Comparison (same type):  $seq < seq$ ,  $seq <= seq$ ,  $seq > seq$ ,  $seq >= seq$ .

# Sequence comparisons

- \* Two sequences are equal if they have the same length and equal elements in every position.
- \*  $seq1 < seq2$  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  $seq2$ .
- \* *Note:* Comparison of NumPy arrays is *element-wise* and returns an array of `bool`.

# 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"`

# Iteration over sequences

# The `for .. in ..` statement

```
for name in expression :  
    suite
```

1. Evaluate the expression, to obtain an iterable collection.
  - If value is not iterable: **TypeError**.
2. For each element  $E$  in the collection:
  - 2.1 assign *name* the value  $E$ ;
  - 2.2 execute the loop suite.

```
for char in "The quick brown fox":  
    print(char, "is", ord(char))
```

**VS.**

```
s = "The quick brown fox"  
i = 0  
while i < len(s):  
    char = s[i]  
    print(char, "is", ord(char))  
    i = i + 1
```



# Iteration over sequences

- \* Sequences are an instance of the general concept of an *iterable* data type.
  - An iterable type is defined by supporting the `iter()` function.
  - python also has data types that are iterable but not indexable (for example, sets and files).
- \* The `for .. in ..` statement works on any iterable data type.
  - On sequences, the `for` loop iterates through the elements *in order*.



# String methods

# Methods

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

```
"Hello World".find("o")
```

instead of

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

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

# Programming problem

- \* Find a longest repeated substring in a word:
  - 'backpack' → 'ack'
  - 'singing' → 'ing'
  - 'independent' → 'nde'
  - 'philosophically' → 'phi'
  - 'monotone' → 'on'
  - 'wherever' → 'er'
  - 'repeated' → 'e'
  - 'programming' → 'r' (or 'g', 'm')
  - 'problem' → ''