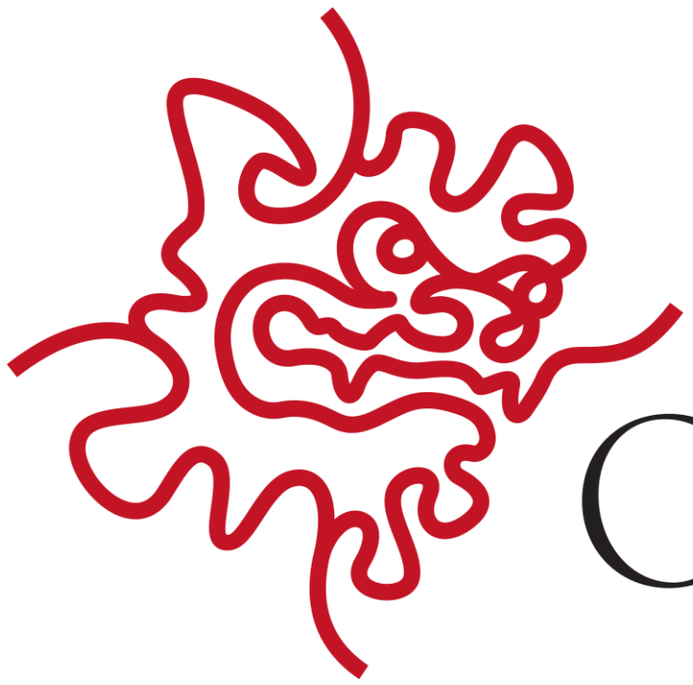
SKILLPILLS

Skill Pill: Intro to Programming

Strings, Control Structures

Kenneth Dudley



OIST

Outline

- Strings
- Control Structures

Strings

- Strings are a way to store text
- Delimited by ‘ (single quote) or “ (double quote)
  - `x = “hello”`
  - `x = ‘this works’`
  - `x = “this ‘works’ too”`
- Strings are immutable iterators

Strings act like lists

- “hello”[0] # returns “h”
- “hello”[1:3] # returns “el”
- “help” in “hello” # returns False
- “hi” + “ ” + “mom” # returns “hi mom”
- len(“hello”) # returns 5
- min(“hello”) # returns “e”
- sorted(“hello”) # returns ['e', 'h', 'l', 'l', ‘o’]
- However, strings cannot be mutated (cannot use append, insert, del…)

Special characters

- Need to be escaped with a `\` (backslash)
  - Tabulation: `\t`
  - New line: `\n`
  - Quotes: `\"`, `\'`
  - Backslash character: `\\`
  - String literal: `r''`

String methods

SKILLPILLS

- `s.upper(), s.lower() # returns upper/lower case characters`
- `s.strip() # returns string without spaces`
- `s.find(s1) # if s1 is inside s, returns the index of s where s1 starts, otherwise -1`
- `s.split(sep) # splits s into chunks separated by sep`
- `sep.join([s1, s2, …]) # joins the strings with separator sep`
- `s.replace(old, new) # replaces occurrences of old with new`
- `“1".zfill(3) # returns “001”`

Format

- `"I like {} and {}." .format("apples", “oranges")`
  `# 'I like apples and oranges.’`
- `"I like {1} and {0}." .format("apples", "oranges")`
  `# ’I like oranges and apples.’`
- `"I like {1}, {0} and {0}." .format("apples", "oranges")  # ’I like oranges, apples and apples.’`
- `“An apple is {price} yen”.format(price=5000)`
  `# 'An apple is 5000 yen’`
- `“Pi is {}”.format(math.pi) # 'Pi is 3.141592653589793’`
- `“Pi is {:f}”.format(math.pi) # 'Pi is 3.141593'`
- `“Pi is {:07.3}”.format(math.pi) # 'Pi is 0003.14’`

Exercise

strings.py

Time: 30 minutes

Control Structures

- We need to be able to make decisions depending on context and repeat actions
- If, else, elif
- While loops
- For loops
- Bonus: list comprehensions

If condition

4 space indentation

Testing condition

Alternative condition

```
age = 17

if age < 16:
    print("Drink soda")
elif age > 19:
    print("Have a beer")
else:
    print("Peket it is!")
```

default action

- We can have many elif
- elif and else are optional
- If conditions can be nested

While loops

```
a=0
while a < 5:
    print(a)
    a += 1
print("Done!")
```

- The indented instructions keep running while the condition is true
- Beware infinite loops (ctrl + c or red square will break the computation)

For Loops

SKILLPILLS

```
for i in range(1,300):
    print(i)
    if i==256:
        print("Yay!")
```

- Loops through the elements of a list (or any iterable)
- Prefer a for loop to a while loop whenever you can!

Unpacking within a loop

SKILLPILLS

```
for x in [(1,2), (3,5)]:
    print(x[0], x[1])


for i, j in [(1,2), (3,5)]:
    print(i, j)
```

- If the elements in the iterable are in a tuple or a list, they can be unpacked directly
- Related functions:
  - `zip([4, 9], "hi") # [(4, "h"), (9, "i")]`
  - `enumerate("hi") # [(0, "h"), (1, "i")]`

Aggregating

```
nums = [2, 49, -7, 0, 14]

sum_nums = 0
for n in nums:
    sum_nums += n
print(sum_nums)

double_nums = []
for n in nums:
    double_nums.append(2*n)
print(double_nums)
```

List Comprehensions

SKILLPILLS

- For loops can be used to create a new list from the elements of another
  - `[i*i for i in range(10)]`
- You can filter the list
  - `[i*i for i in range(10) if i!=5]`
- You can nest loops
  - ```
    [(i, j) for i in range(10)
            for j in range(10,20)]
    ```


    ```

Exercise

loops.py

while time<end:
code()

Bonus

- More Fun with Strings - FYI
  - Regex – Regular Expression
  - BonusRegex.py

Regular Expression

SKILLPILLS

- Regex is a powerful tool to match complex combinations of patterns
- Nearly all languages have a Regex implementation:

  Python/Matlab/C++/R/Julia/Bash/Java/…
- Implementations are mostly Identical

Simple Example

```
import re

first_example = 'simple text with the number 47 in it'

reg_pat = r'[0-9]+'

reg_first_result = re.findall(reg_pat, first_example)

print('\tSimple Result\n\t\t',reg_first_result)
```

Why Regex

SKILLPILLS

- Find multiple matches
- Less code
- Potentially simpler logic
- Match multiple conditions at once

  Find a word after number that does not contain a 5 in a string.

"Find 259 that 111 **word** behind behind"

Without Regex

SKILLPILLS

```
second_example = "Find 259 that 111 word behind behind"

split_str = second_example.split()

no_5 = [i.isnumeric() if '5' not in i else False for i in split_str]

idx_num = no_5.index(True)

the_word = split_str[idx_num + 1]

print('\tNot Regex:\n\t\t', the_word)
```

With Regex

SKILLPILLS

```
second_example = "Find 259 that 111 word behind behind"

import re

reg_str = r'(?<!\d)(?:[0-46-9]+)(?!\d)\W*(\w+)'

reg_second_result = re.findall(reg_str, second_example)

print('\tYes Regex:\n\t\t', reg_second_result)
```

Can be very complex

SKILLPILLS

```
ambig_parse_re = re.compile(r"""\A
   # Latitude direction, first position: one of N, S, NORTH, SOUTH
   #Check for negative postive indicators of decimal degrees
   (?P<errlat>(\D+)\ ?)?

   #Check northings eastings 4+ digits
   (?P<latnthest>(\d{4,}))?

   # Check for DMm where minutes are given with decimal minutes instead of seconds
   (?P<latDMm>((?<!\d\.)(?<!\d)[0-9]{1,2}(?!\d) (º|°) (\.|\s)? (60|[0-5]?[0-9])(?![\dº°]) (\.\d+) (''|'|")? (?!\d) ))?

   # Check for splits on decimals with D.M.S.ss
   (?P<latDMSs>((?<!\d\.)(?<!\d)[0-9]{1,2}(?!\d)\.[0-9]{1,2}(?!\d)\.[0-9]{1,2}(?!\d)(\.\d+)?))?

   # Check for standard DMS but probably without NSEW or possibly corrupted text
   (?P<latDMS>((?<!\d\.)(?<!\d)[0-9]{1,2}(?!\d) (?P<degmark>(º|°))? (?P<spacer>\.|\ )? (60|[0-5]?[0-9])(?![\dº°]) (''|'|")? (?P=spacer)? ([0-9]{1,2}(?![\dº°])(\.\d+)? (''|'|')?)? (?!\.?\d) ))?
#    (?P<latDMS>([0-9]{1,2}(?!\d) (?P<degmark>(º|°))? (\.|\s)? (60|[0-5]?[0-9])(?![\dº°]) (\s)? (''|'|")? (\s)? (\d+(?![º°])(\.\d+)? (''|'|')?)? (?!\.?\d) ))?

   # Check for DD in DMS format D.mssss Three or more digits after decimal
#    (?P<latDDinDMS>([0-9]{1,2}(?!\d)\.\d+\s?(?![\.º°])))?
   (?P<latDDinDMS>((?<!\d\.)(?<!\d)[0-9]{1,2}(?!\d)\.\d+(?!\d?\.\d)(?!\d?\s?[º°])))?

   # Check for missplaced degree mark
   (?P<latBadDeg>(?<!\d\.)(?<!\d)[0-9]{1,2}(?!\d)\.\d+(\ ?)(º|°))?

   #--------------#
   (\s+)?[\\/\|,]?
   #--------------#
   # Latitude direction, second position: one of N, S, NORTH, SOUTH
   #do not capture - + markers if they exist after the first coordinate as the ending of that first coordinate, it likely refers to the next coordinate
#    (?(errlat) | (?P<errlat2>([\D](?![-\+]) )) )?
   (?(errlat) | (?![-\+])(?P<errlat2>([\D] )) )?

   #--------------#
   # Latitude/longitude delimiter: space, forward slash, comma, or none
   (\s+)?[\\/\|,]?(\s+)?
   #--------------#

   # Longitude direction, first position: one of E, W, EAST, WEST
   ((?P<errlon>\D+))?(\s?)

   #Check northings eastings 4+ digits
   (?P<lonnthest>(\d{4,}))?

   # Check for DMm where minutes are given with decimal minutes instead of seconds
   (?P<lonDMm>((?<!\d\.)(?<!\d)[0-9]{1,3}(?!\d) (º|°) (\.|\s)? (60|[0-5]?[0-9])(?![\dº°]) (\.\d+) (''|'|")? (?!\d) ))?
```