Python(3) Cheat Sheet

Instructor: Farbod Parvin

Command Prompt Basics

- Accessing another drive:
  - from "C:" to "D:“ ☞ type ‘`d:`’

```
C:\Users\Farbod>d:

D:\>
```

- Changing the directory:
  - from “C:” to “C:\windows\system32\” ☞ type ‘`cd windows\system32`’

```
C:\>cd windows\system32

C:\Windows\System32>
```

- Going one folder up:
  - from “C:\windows\system32\” to “C:\windows\” ☞ type ‘`cd..`’

```
C:\>cd windows\system32

C:\Windows\System32>
```

Command Prompt Basics (Python)

- Running python interpreter: ☞ type ‘`python`’
- Exiting interpreter: ☞ type ‘`quit()`’
- Running a python script: ☞ type ‘`python my_file.py`’
  - Note: you should be in the file directory!!!

Numbers

- Types:
  -

Name	Type	Example
Integer	`int`	`21  100  -5`
Floating point	`float`	`21.0  42.32  0.12`
Complex	`complex`	`2+3j 100+0j`

- Type Conversion:
  - `2 + 3.0` ☞ returns `5.0`
  - `2 + int(3.0)` ☞ returns `5`
  - `3 / 2` ☞ returns `1.5`
  - `3 // 2` ☞ returns `1`

Operators			
Sum `+`	Subtract `-`	Multiply `*`	Division `/`
Floor division `//`	Mod `%`	Power `**`	

Variables

- Definition: a name attached to a particular object
- Python is a dynamically-typed language (as opposed to a statically-typed language)
- Dynamically-typed: type of variables need **not** be declared or defined in advance
- Give a variable a name that is descriptive enough to make clear what it is being used for
- Naming conventions:
  - uppercase and lowercase letters (A-Z, a-z)
  - digits (0-9)
  - underscore character (_)
  - Snake Case: Words (lower-cased) are separated by underscores
    - Example: number_of_iterations = 20

Strings

- Creation: `my_str = “Hello World”` or `‘Hello World’`
- Printing: `print(my_str)` ☞ returns `“Hello World”`
- Print formatting: ☞ returns `‘inject text: hello and world’`
  - Old: `‘inject text: %s and %s’ % (‘hello’, ‘world’)`
  - New: `‘inject text: {} and {}’.format(‘hello’, ‘world’)`
- Indexing: `my_str[1]` ☞ returns `‘e’` (zero-based index)
  - My_str[-1] ☞ returns ‘d’

	H	**e**	**l**	**l**	**o**
ind	0	1	2	3	4

- Slicing: my_str[1:4:1] ☞ returns ‘ell’ (grab from index 1 up to, but not including 4, with the step of 1)
- Length: len(my_str) ☞ returns 11
- Concatenation: `my_str + ‘ farbod’` ☞ returns `“Hello World farbod”`
- Repetition: `‘a’*10` ☞ returns `‘aaaaaaaaaa’`

Formatted output

- Old way: ‘my name is %s and I am %d years old’ % (‘farbod’, 26)
- *Pythonic* way:
  - `‘my name is {} and I am {} years old’.format(‘farbod’, 26)`
  - OR `‘my name is {esm} and I am {sen} years old’.format(esm=‘farbod’,sen=26)`
  - OR `‘my name is {1} and I am {0} years old’.format(26, ‘farbod’)`

index = 0	1

- Ver. 3.6 and newer:
  - `esm = ‘farbod’`
  - `sen = ‘26’`
  - `f‘my name is {esm} and I am {sen} years old’`

Strings

- IMPORTANT:
- Strings are <u>immutable</u>: Once it is created, you cannot change the contents of the string.
  - `a = ‘Hello’, a[1] = ‘o’` ☞ returns error
- Methods: Functions that are inside an object and use that object as input.
- String methods:
  - `a.upper()` ☞ returns `‘HELLO’`
  - `a.lower()` ☞ returns `‘hello’`
  - `a = ‘farbod parvin’, a.split()` ☞ returns `[‘farbod’,‘parvin’]`

    `a.split(‘ar’)` ☞ returns `[‘f’,’bod p’,’vin’]`

Commenting

- Single line comment (#):
  - `# tozihaat dar yek khat`

- Multi-line comment (‘’’)
  - ‘’’ tozihaat be sharh e zir ast:
    khate 1
    khate 2 ‘’’

Strings (indexing and slicing)

	H	**e**	**l**	**l**	**o**
ind	0	1	2	3	4
ind	-5	-4	-3	-2	-1

- `my_str = “Hello”`
- `my_str[1:4]` 👉 `“ell”` | `my_str[:4]` 👉 `“Hell”` | `my_str[1:]` 👉 `“ello”`
- `my_str[:-3]` 👉 `“He”` | `my_str[-3:]` 👉 `“llo”`
- Reversing: `my_str[::-1]` 👉 `“olleH”`

Booleans, comparison operators and logical operators

Booleans	
`True`	`False`

Comparison operators					
Equal to	Not equal to	Greater than	Less than	Greater than or equal to	Less than or equal to
`==`	`!=`	`>`	`<`	`>=`	`<=`

Logical operators		
`and`	`or`	`not`

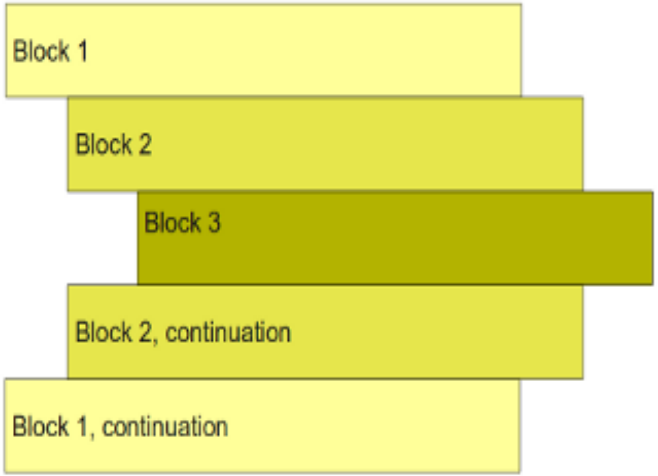
Booleans, comparison operators and logical operators

- Boolean definition: `a = True` OR `b = False`
- Checking for equality and inequality: `car = 'bmw'`
  - `car == 'bmw'` ☞ returns `True`
  - `car == 'audi'` ☞ returns `False`
  - `car != 'audi'` ☞ returns `True`
  - `not car == 'bmw'` ☞ returns `False`
- Checking multiple conditions: `age_1, age_2 = 22, 18`
  - `age_1>=21 and age_2>=21` ☞ returns `False`
  - `age_1>=21 or age_2>=21` ☞ returns `True`

Conditional Tests	**equal**	**not equal**	**greater than (or equal to)**	**less than (or equal to)**
	x == 42	**x != 42**	**x > (>=) 42**	**x < (<=) 42**

Indentation and If statement

- A logical block of statements such as ‘if’ statements should all have the same indentation



-
```
if age >= 18:
    print(“you’re old enough to vote”)
else:
    print(“you can’t vote yet”)
```

Indentation = 1 Tab or 4 spaces

Indentation and if statement

If statement keywords		
`if`	`elif`	`else`

-
```
if speed >= 80:
    print 'License and registration please'
    if mood == 'terrible' or speed >= 100:
        print 'You have the right to remain silent.'
    elif mood == 'bad' or speed >= 90:
        print "I'm going to have to write you a ticket."
    else:
        print "Let's try to keep it under 80 ok?“
```