# CPSC 231 Tutorial #13

michael-hung.ca/teaching

# Reminders

---

**TODAY**

Quiz 7

**THURSDAY**

Quiz 7 Review

**FRIDAY**

Assignment 4 Individual Component Due

**OCTOBER 25**

Alberta Collegiate Programming Contest (ACPC) Signup Deadline
*Sign up for **Division 2** since you haven't taken CPSC 319/331*

# String Things

---

A string is sort of like an array of characters:

“Hello” ⇔ [‘H’, ‘e’, ‘l’, ‘l’, ‘o’]

We can get a character at a specific index with [], e.g. s[0] ⇒ “H”

**HOWEVER!**

Unlike arrays, a string is **immutable**. You *cannot* change individual characters like you would an array element.

```
s = “Hello”
s[0] = “J”    # This will throw a TypeError
```

# String Operations (Review)

---

**s**.strip()

Returns **s** with all leading and trailing whitespace removed

**s**.split(*d*)

Returns an array of words in **s**, separated by delimiter *d* (all occurrences of *d* are also removed)

# More String Operations

---

**s**.lower(), **s**.upper()

Returns **s**, converted into all lowercase or uppercase

**s**.startswith(*substring*)

If **s** begins with the *substring*, returns True. False otherwise. **Case sensitive!**

**s**.endswith(*substring*)

If **s** ends with the *substring*, returns True. False otherwise. **Case sensitive!**

# More String Operations

---

**s**.find(*substring*)

Returns the index of the **first** occurrence of *substring* in **s**. -1 if not found.

**s**.rfind(*substring*)

Returns the index of the **last** occurrence of *substring* in **s**. -1 if not found.

# The “in” keyword

---

If you only need to know if a string contains a substring, use the **in** keyword, e.g.

```
print(“H” in “Hello”) # prints True

print(“h” in “Hello”) # case sensitive! Prints False

print(“J” in “Hello”) # prints False
```

# More String Operations

---

**s**.replace(*old*, *new*)

Replaces all occurrences of the substring *old* with the substring *new*

# String Formatting

---

Up until now, you've probably been doing:

```
print("Hello", name)
```

or

```
print("Hello " + name)
```

# String Formatting

---

**s**.format(arg0, arg1, ...)

Use curly braces inside the string to indicate placeholders that correspond to the number of arguments, e.g.

```
firstName = “Alice”
lastName = “Brown”
s = “My name’s {0} {1}!”.format(firstName, lastName)
print(s) # Prints “My name’s Alice Brown.”
```

# String Formatting (printf-style)

---

```
name = “Alice”
s2 = “Hello %s” % (name)

numOfDice = 2
avg = (random.randrange(1,7) + random.randrange(1,7)) / 2

s1 = “Rolled %d dice for an average of %f.” % (numOfDice, avg)
print(s1) # Prints “Rolled 2 dice for an average of 3.500000.”
```

# String Formatting (printf-style)

---

| CONVERSION | MEANING |
| --- | --- |
| d | Integer |
| f | Floating Point |
| s | String, or tries to convert object with **str()** |

# String Slicing

| SYNTAX | RETURNS | EXAMPLE (s = “abcdef”) |
|---|---|---|
| s[start:end] | Substring from start index to end index, exclusive | s[0:3] ⇒ “abc” |
| s[start: ] | Substring from start index to end of string | s[3:] ⇒ “def” |
| s[ :end] | Substring from beginning of string to end index, exclusive | s[:2] ⇒ “ab” |
| s[start:-n] | Substring from start to the nth-last index | s[1:-1] ⇒ “bcde”<br>s[-3:-1] ⇒ “de” |
| s[start:end:n] | Substring from start to end for every nth character | S[::2] ⇒ “ace”<br>S[::-1] ⇒ “fedbca” |

# Anagram Checker

---

Given two strings, **string1** and **string2**, return True if string2 is an anagram of string1, i.e. string2 has exactly the same letters as string 1. Assume there's neither punctuation or whitespace.

**INPUT**

    string1
    string2

**OUTPUT**

    True        if string2 is an anagram of string1
    False       otherwise

# Anagram Checker Challenge

---

Don’t use array functions this time!

Hint: Convert between characters and their ASCII values.

**ord(***char***)**
    Returns integer value of a single character,
    e.g. ord(‘a’) ⇒ 97

**chr(***asciiValue***)**
    Returns a single character based on its ASCII value,
    e.g. chr(97) ⇒ ‘a’