Strings

MAD 102

Strings

- Strings are a sequence type - they are collections of characters.
- The characters are ordered from left to right
- An index value represents the position of each character in the string
  - Characters can be accessed by placing the index value in between square brackets

```
# Strings Examples
word = 'onomatopoeia'
first_letter = word[0]
print(first_letter)
```

This will get the letter o

Strings - slicing

- If you want to access characters in a string, you can use the slice notation to achieve this
- Slice notation has this form **word[startIndex: endIndex]**
  - This will slice the string and make a new string, starting at the start index position going to, but not including, the end index position

```
# Strings Examples
word = 'onomatopoeia'
upper = word[5:8]
print(upper)
```

This will get the letter t – o - p

Strings - slicing

- You can get the last letter in a string, but going one index beyond the end of the string

```
word = 'batman'
person = word[3:6]
print(person)
```

The last letter is index position 5 – one past is 6
This will return the string ‘man’

Strings - slicing

- Negative numbers can be used to represent the end

```
word = 'batman'
letter = word[-4:-3]
print(letter)
```

This would return the letter ‘t’
Starting at the end going back to the fourth last letter & ending at the third last (but not including).

String - slicing

- If index values are omitted – the beginning and end are assumed

```
word = 'batman'
letter = word[3:]
print(letter)
```

Returns ‘man’

```
word = 'batman'
letter = word[:3]
print(letter)
```

Returns ‘bat’

String - slicing

- The slice notation can have a third argument that represents the **stride**
  - The stride determines how to increment each index value
    - The default is a stride of 1

```
# Strings Examples
word = 'batman'
letter = word[::2]
print(letter)
```

Slices the string, from start to finish – at every 2nd index value – 0, 2, 4,…
This will return b-t-a

Format Specification - Field width

- Keeping string values formatted properly for outputting can be accomplished with the **field width**
- This defines the min. number of characters that must be inserted into a string to maintain a specified formatting
  - If the values are less than the size of the given field, spaces are added
  - Numbers will be right-aligned
  - Any other type will be left-aligned

Format Specification - Field width

- Defined as part of the string format specification

```
# Strings Examples
print(f'{"Student Name":20}{"Grade":10}')
print('='*30)
print(f'{"Malcolm Reynolds":20}{83: 10}')
print(f'{"Zoe Washburne":20}{99:10}')
print(f'{"Jayne Cobb":20}{34:10}')
```

```
Student Name        Grade
==============================
Malcolm Reynolds            83
Zoe Washburne               99
Jayne Cobb                  34
```

Format Specification - Alignment

- Alignment can be adjusted using the **alignment character**
  - Left-aligned <
  - Right-aligned
  - Centered ^

```python
# Strings Examples
print(f'{"Student Name":20}{"Grade":^10}')
print('='*30)
print(f'{"Malcolm Reynolds":20}{83: ^10}')
print(f'{"Zoe Washburne":20}{99:^10}')
print(f'{"Jayne Cobb":20}{34:^10}')
```

```
Student Name           Grade
==============================
Malcolm Reynolds         83
Zoe Washburne            99
Jayne Cobb               34
```

Format Specification – Fill Character

- The fill character is used to fill (pad) the extra space when a string is less than the field width
  - By default, this is spaces.
  - Can be used in conjunction with the alignment character to determine if the fill characters appear before (>), after (<) or around (^)

Format Specification – Fill Character

```python
# Strings Examples
print(f'{"High Scores": ^16}')
print('='*16)
print(f'{"Name":10}{"Score":^6}')
print('='*16)
print(f'{"CDT":10}{830:0>6}')
print(f'{"HBC":10}{645:0>6}')
print(f'{"STN":10}{34:0>6}')
```

```
  High Scores
================
Name      Score
================
CDT       000830
HBC       000645
STN       000034
```

Format Specification - Numeric Precision

- The format specification has a **precision** component that will determine the number of digits to use for your output
  - Starts with a .
  - Followed by the number of digits of precision

```
# Strings Examples
print(f'{math.pi}')
print(f'{math.pi:.2f}')
```

```
3.141592653589793
3.14
```

String Methods

- The **replace()** method is used to find an occurrence of a substring with a new substring
  - Strings are immutable – editing one string means a new string will be created and returned
  - The replace method has two arguments – one representing the substring to find, the other the substring to replace.

String Methods

```python
# Strings Examples
title = 'The new adventures of Indiana Jones'

print(title.replace('new', 'continuing'))
```

```
/Users/dtakaki/PycharmProjects/example/venv/bin
The continuing adventures of Indiana Jones
```

String Methods

- The replace also contains a **count** argument that will replace only that many instances of the substring

```
# Strings Examples
phrase = 'This phrase is very, very, very, very long'

print(phrase.replace('very, ', '', 2))
```

Count

```
This phrase is very, very long
```

String Methods

- The **find()** method can be used to determine the index position of where a substring starts – this is returned as an integer value

```
title = 'Indiana Jones and the Dial of Destiny'

print(title.find('Dial'))
```

```
22
```

String Methods

- If the string is substring is not found it returns -1

```
2  title = 'In a galaxy far, far away'
3
4  print(title.find('the'))
5
```

```
-1
```

- Where there are multiple – it returns the **first** occurrence

```
2  title = 'In a galaxy far, far away'
3
4  print(title.find('far'))
5
```

```
12
```

String Methods

- **rfind(x)** method – starts searching from the end

```
2   title = 'In a galaxy far, far away'
3
4   print(title.rfind('far'))
5
```

```
17
```

- **find(x, startPosition)** will start searching from the designated start position index
- **find(x, startPosition, endPosition)** will perform the search between a starting and ending position

String Methods

- **count(x)** method will return the number of times a substring appears in a string

```
2   title = 'In a galaxy far, far away'
3
4   print(title.count('far'))
5
```

```
2
```

Comparing Strings

- Strings can be compared using:
  - relational operators (<, <=, >=, >)
  - equality operator (==, !=)
  - Membership operators (in, not in)
  - Identity operators (is, is not)
- Comparison uses the encoded values of the characters (ASCII/Unicode)

Comparing Strings - methods

- **isalnum()** – returns True if all characters are lowercase or uppercase letters or numbers 0-9
- **isdigit()** – returns True is all characters are numbers 0-9
- **islower()** – returns True if all characters are lowercase
  - **isupper()** if all characters are uppercase
- **isspace()** – returns True if all characters are whitespace
- **startswith(x)** – returns True if it starts with this character
- **endswith(x)** – returns True if it ends with this character

Creating new Strings

- The following methods create new strings
- **capitalize()** – returns a copy of the string with the first character capitalized
- **lower()** or **upper()** will lowercase or uppercase the string
- **strip()** removes any leading or trailing whitespace
- **title()** returns a string with the first letter of every word capitalized

Creating new Strings

Methods can be chained together

```
2   title = '    In a galaxy far, far away '
3
4   print(title.strip().title())
5
```

```
In A Galaxy Far, Far Away
```

Remove the trailing and leading whitespace
Capitalize the first letter of each word

Creating new Strings

- Good practice is to apply transformations when string values are being read in
  - This is a way of cleaning user input, removing any errors that the user might have implemented
    - i.e typing n9a 6S4 instead of N9A 6S4

Splitting Strings

- The **split()** method splits a string into a **list** of **tokens**.
  - Each **token** is a substring
  - A **separator** is a character (or sequence of characters) that indicates where to split the string into a **list** of tokens
  - By default – the split uses a blank space as the separator

```
2   title = 'In a galaxy far, far away'
3
4   print(title.split())
5
```

```
['In', 'a', 'galaxy', 'far,', 'far', 'away']
```

Splitting Strings

- Define the separator by using it as an argument for the **split()** method

```
2  title = 'In a galaxy far, far away'
3
4  print(title.split(','))
5
```

```
['In a galaxy far', ' far away']
```

Joining Strings

- The **join()** method joins a list of strings around a designated separator

```
2  url = ['https:/', 'dtakaki.scweb.ca', 'MAD102', 'intro']
3  separator = '/'
4  print(separator.join(url))
5
```

```
https://dtakaki.scweb.ca/MAD102/intro

Process finished with exit code 0
```

Strings

- To get the number of characters in a string value, use the **len()** method

```
2   title = 'In a galaxy far, far away'
3
4   print(len(title))
5
```

```
25
```