loops + strings (class slides)

CSc 110 - looping through strings

String Manipulation

String indexing

- Strings are sequences in Python
- We can retrieve values in a sequence using `[ ]`

```
name = "Adriana"
name[0]
```

```
'A'
```

Note that the first index in a sequence is always `zero`.

String indexing

Since the first index of a sequence is `zero`, the last index is going to be the length of the string `minus 1`

```
name = "Adriana"
len(name)
```

```
7
```

```
name = "Adriana"
name[6]
```

```
'a'
```

```
name = "Adriana"
name[7] # this will throw an error
```

Evaluate expressions

```
my_str = "Clint Eastwood"
my_str[11] + my_str[1] + my_str[13]

my_str = "cider"
my_str[0] + my_str[4] + my_str[1] + my_str[3] + my_str[2]

my_str = "dusty"
my_str[2] + my_str[3] + my_str[1] + my_str[0] + my_str[4]
```

Evaluate expressions

```
my_str = "Clint Eastwood"
my_str[11] + my_str[1] + my_str[13]
```

```
'old'
```

```
my_str = "cider"
my_str[0] + my_str[4] + my_str[1] + my_str[3] + my_str[2]
```

```
'cried'
```

```
my_str = "dusty"
my_str[2] + my_str[3] + my_str[1] + my_str[0] + my_str[4]
```

```
'study'
```

Write a function

1. Its name is `four_letter_anagram`
2. Its parameters are a `string`, and four integers `a`, `b`, `c` and `d`
3. It returns the anagram of the string according to the indices `a`, `b`, `c` and `d` – concatenate the individual characters
4. Test case: `four_letter_anagram("balm", 2, 1, 3, 0)` should return `lamb`

Write a function - solution + add to it

Add the following test cases to the code below:

- `loin` returns `lion`
- `lugs` returns `slug`
- `reap` returns `pear`

```
def four_letter_anagram(word, a, b, c, d):
  return word[a] + word[b] + word[c] + word[d]

def main():
  print( four_letter_anagram("balm", 2, 1, 3, 0) )

main()
```

```
lamb
```

Submit code for attendance

Submit your `four_letter_anagram` function to Gradescope for attendance.

Name your file `anagram.py`

`in` operator

The `in` operator determines whether a given value is a constituent element of a sequence (such as a string)

```
"a" in "Adriana"
```

```
True
```

```
"ana" in "Adriana"
```

```
True
```

```
"i" in "aeiou"
```

```
True
```

Evaluate the expressions

```
"a" in "aeiou"
"b" in "aeiou"
not "b" in "aeiou"
"0" in "0987654321."
"10.0" in  "0987654321."
```

Evaluate the expressions

```
"a" in "aeiou"
```

```
True
```

```
"b" in "aeiou"
```

```
False
```

```
not "b" in "aeiou"
```

```
True
```

```
"0" in "0987654321."
```

```
True
```

```
"10.0" in  "0987654321."
```

```
False
```

Before we proceed, a refresher ...

Remember the string method `.isnumeric()`?

```
user_input = "paul"
user_input.isnumeric()
```

```
False
```

```
user_input = "12"
user_input.isnumeric()
```

```
True
```

```
user_input = "12.5"
user_input.isnumeric()
```

```
False
```

Write your own `is numeric` for one character

Write a Python function that does the following:

1. Its name is `is_numeric_one_char`
2. It takes one `string` argument of length of 1 (check if argument is valid with `len()`)
3. It returns `True` if the argument is a digit (0-9), `False` otherwise (use the `in` operator)

```
print( is_numeric_one_char("0") ) # True
print( is_numeric_one_char("000") ) # False
print( is_numeric_one_char("9") ) # True
print( is_numeric_one_char("a") ) # False
```

Write a function – solution

```
def is_numeric_one_char(char):
  if len(char) > 1:
    return False

  return char in "0123456789"

def main():
  print( is_numeric_one_char("0") ) # True
  print( is_numeric_one_char("000") ) # False
  print( is_numeric_one_char("9") ) # True
  print( is_numeric_one_char("a") ) # False

main()
```

```
True
False
True
False
```

Write a `is_numeric()` for any string length

1. Its name is `is_numeric`
2. It takes one `string` argument of any length
3. It returns `True` if the **first character** of the argument is a digit (0-9), `False` otherwise (remember to use `[ ]` to index the string and the `in` operator)

Test cases:

```
print( is_numeric("0") ) # True
print( is_numeric("000") ) # True
print( is_numeric("9") ) # True
print( is_numeric("a") ) # False
```

Write a function – solution

```
def is_numeric(char):
  return char[0] in "0123456789"

def main():
```

```
  print( is_numeric("0") ) # True
  print( is_numeric("000") ) # True
  print( is_numeric("9") ) # True
  print( is_numeric("a") ) # False

main()
```

```
True
True
True
False
```

Improving on `is_numeric()`

What if we want to check every character in a string of any length?

What if the string is of length `2` or `5` or `45`?

While loops – using an index

- One technique that can be used to control the number of loop iterations is using an index variable
- For while loops, an index variable is:
  - Defined before the loop
  - Used in the condition of the loop
  - Incremented within the loop

While loops – using an index

```
index = 0
while index < 5:
    print('Print ' + str(index))
    # Can add other lines here too
    index = index + 1
```

```
Print 0
Print 1
Print 2
```

```
Print 3
Print 4
```

```
index = 5
while index > 0:
    print('Print ' + str(index))
    # Can add other lines here too
    index = index - 1
```

```
Print 5
Print 4
Print 3
Print 2
Print 1
```

Improve on your `is_numeric()`

1. Its name is `is_numeric`
2. It takes one `string` argument of any length
3. It returns `True` if every character in the argument is a digit (0-9), `False` otherwise
4. Remember to use `[ ]` to index the string, use a while loop with a `len()` condition, and an index – something like `string[index]` with index being updated inside a `while` loop

```
print( is_numeric("0") ) # True
print( is_numeric("1090") ) # True
print( is_numeric("95") ) # True
print( is_numeric("10a") ) # False
```

`is_numeric()` solution

```
def is_numeric(my_string):
  index = 0
  while index < len(my_string):
    if my_string[index] not in "0123456789":
      return False
    index += 1
  return True

def main():
```

```
  print( is_numeric("234") ) # True
  print( is_numeric("abc") ) # False
  print( is_numeric("12c") ) # False
  print( is_numeric("12.3") ) # False

main()
```

```
True
False
False
False
```

`is_numeric()` solution

How do we change our function so that the last test case ("12.3") returns `True` instead?

```
def is_numeric(my_string):
  index = 0
  while index < len(my_string):
    if my_string[index] not in "0123456789":
      return False
    index += 1
  return True

def main():
  print( is_numeric("234") ) # True
  print( is_numeric("abc") ) # False
  print( is_numeric("12c") ) # False
  print( is_numeric("12.3") ) # False

main()
```

```
True
False
False
False
```

`is_numeric()` solution

How do we change our function so that the last test case ("12.3") returns `True` instead?

```
def is_numeric(my_string):
  index = 0
  while index < len(my_string):
    if my_string[index] not in "0123456789.":
      return False
    index += 1
  return True

def main():
  print( is_numeric("234") ) # True
  print( is_numeric("abc") ) # False
  print( is_numeric("12c") ) # False
  print( is_numeric("12.3") ) # True
  print( is_numeric("1.2.3") ) # True

main()
```

```
True
False
False
True
True
```

`is_numeric()` solution

```
def is_numeric(my_string):
  # create control variable for one decimal point
  decimal_point = False
  # create index variable
  index = 0
  while index < len(my_string):
    # first check, can character be found in a number?
    if my_string[index] not in "0123456789.":
      return False
    # second check, if a period, has a period been found before?
    if my_string[index] == ".":
      if decimal_point: # a previous period was detected
        return False
      else: # first period detected
        decimal_point = True
```

```
    # increment index
    index += 1
  # while loop executed without returning False
  # that means every character is valid, so return True
  return True

def main():
  print( is_numeric("234") ) # True
  print( is_numeric("abc") ) # False
  print( is_numeric("12c") ) # False
  print( is_numeric("12.3") ) # True
  print( is_numeric("1.2.3") ) # False

main()
```

```
True
False
False
True
False
```

Quiz 04

current time

You have 10 minutes to complete the quiz

- No need for comments, no need for a `main()`, no need to print test cases
- Just write your function and what's inside the function
- DO NOT USE NESTED CONDITIONALS

Built-in functions you can use: `round()`, `input()`, `float()`, `str()`, `int()` — you don't have to use all of these