CSc 110 Files, Strings and Debugging

Benjamin Dicken

What is a palindrome?

- A palindrome is a string that reads the same both forwards and backwards. Some examples of palindromes:

```
'civic'
'radar'
'rotator'
```

Write the function is_palindrome_word

- A palindrome is a string that reads the same both forwards and backwards. Some examples of palindromes:
  - civic, radar, rotator

- `is_palindrome_word('civic')` should return `True`
- `is_palindrome_word('non')` should return `True`
- `is_palindrome_word('contemporary')` should return `False`

```python
def is_palindrome_word(word):
    forwards = word
    backwards = ''
    i = len(word) - 1
    while i >= 0:
        backwards = word[i] + backwards
        i -= 1
    return forwards == backwards

b1 = is_palindrome_word('contemporary')
print(b1)
b2 = is_palindrome_word('civic')
print(b2)
```

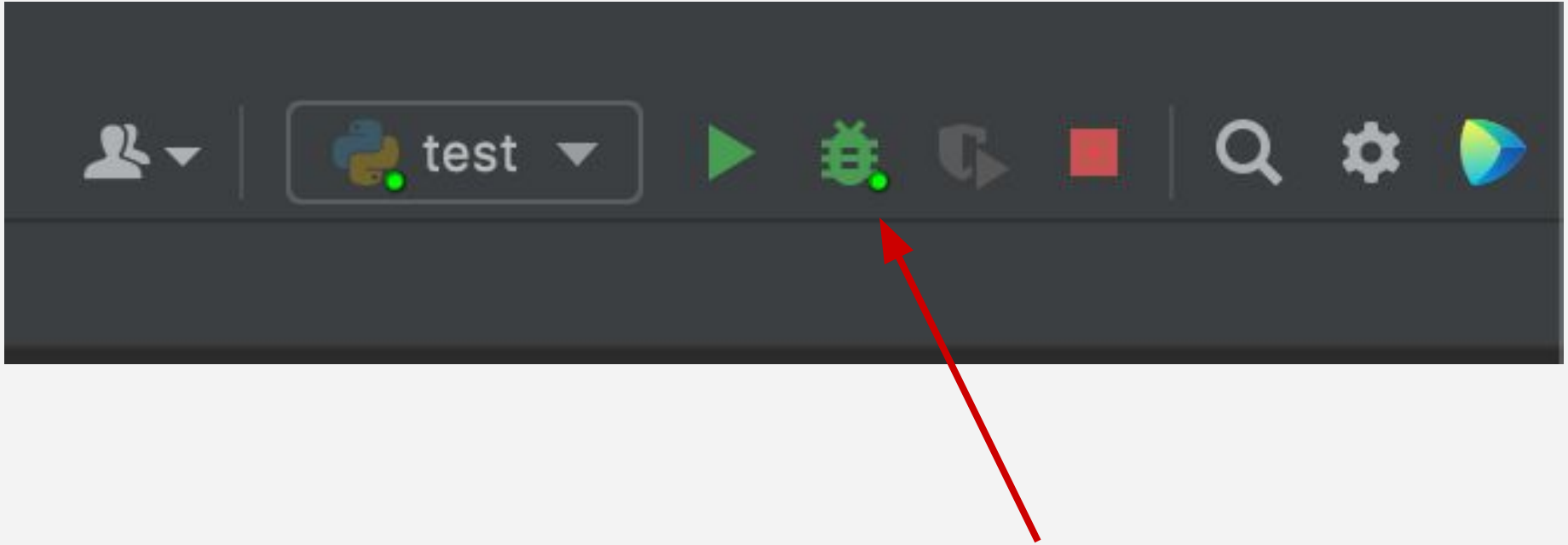
Activity

```python
def is_palindrome_word(word):
    forwards = word
    backwards = ''
    i = len(word) - 1
    while i >= 0:
        backwards = word[i] + backwards
        i -= 1
    return forwards == backwards
```

Is this correct?

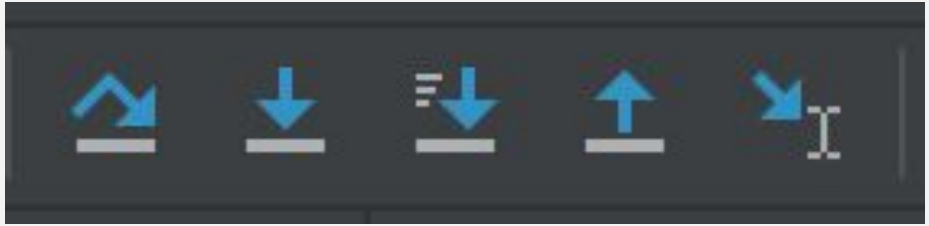
```python
b1 = is_palindrome_word('contemporary')
print(b1)
b2 = is_palindrome_word('civic')
print(b2)
```

The debugger . . . have you used it ?



The debugger . . . have you used it ?

- First, set a “breakpoint” (red dot) and then you can:
- **Step over:** step over (move past) the current line (highlighted line)
- **Step in:** follow a function call on the current line
- **Step out:** Step out of the currently executing function (not sure it it works right though)



```
1   def is_palin
2       forwards
3       backward
4       i = len(
5       while i
6           back
7           i -=
8       return f
9
10  b1 = is_pali
11  print(b1)
12  b2 = is_pali
13  print(b2)
14
```

```python
def is_palindrome_word(word):
    forwards = word
    backwards = ''
    i = len(word) - 1
    while i >= 0:
        backwards = backwards + word[i]
        i -= 1
    return forwards == backwards

b1 = is_palindrome_word('contemporary')
print(b1)
b2 = is_palindrome_word('civic')
print(b2)
```

```python
def is_palindrome_word(word):
    forwards = word
    backwards = ''
    i = len(word) - 1
    while i >= 0:
        backwards = backwards + word[i]
        i -= 1
    return forwards == backwards

print(is_palindrome_word('radar'))
print(is_palindrome_word('hi there'))
```

Try to do this with minimal number of lines

```python
def is_palindrome_word(word):
    i = 0
    while i < len(word):
        if word[i] != word[len(word) - 1 - i]:
            return False
        i += 1
    return True


print(is_palindrome_word('radar'))
print(is_palindrome_word('hi there'))
```

```python
def is_palindrome_word(word):
    for i in range(len(word)):
        if word[i] != word[len(word) - 1 - i]:
            return False
    return True


print(is_palindrome_word('radar'))
print(is_palindrome_word('hi there'))
```

```python
def is_palindrome_word(word):
    return word == word[::-1]
```

Write the function is_palindrome

Ignore spaces

- `is_palindrome('otto sees otto')` should return `True`
- `is_palindrome('olson is in oslo')` should return `True`
- `is_palindrome('radar')` should return `True`
- `is_palindrome('one two three')` should return `False`
- `is_palindrome('three')` should return `False`

```python
def is_palindrome(string):
    string_split = string.split()
    string = ''
    for i in string_split:
        string = string + i
    i = 0
    while i < len(string):
        if string[i] != string[len(string) - 1 - i]:
            return False
        i += 1
    return True
```

```python
def is_palindrome(string):
    string_split = string.split()
    string = ''
    for i in string_split:
        string = string + i
    return is_palindrome_word(string)
```

```python
def is_palindrome(string):
    string_split = string.split()
    string = ''.join(string_split)
    return is_palindrome_word(string)
```

What does join do?

- The **`join()`** function is a built-in function (method) for strings
- Usage:

```
string.join(list)
```

- Concatenates all of the elements in **`list`**, putting **`string`** in-between each element
- Can think of it as the reverse of what split does

What does join do

```
sentence = 'What is your name?'

s_1 = sentence.split(' ')      # ['What', 'is', 'your', ‘name?']

s_2 = ' '.join(s_1)            # 'What is your name?'
```

What will it print?

```
sample = 'the finest of the wheat and wine'
s1 = sample.split(' ')
s2 = s1[1:4]
s3 = '--'.join(s2)
s4 = s3.split('-')

print(s3)
print(s4)
```

```python
def is_palindrome(string):
    string_split = string.split(' ')
    string = ''.join(string_split)
    return is_palindrome_word(string)
```

Write the function most_used_char

- The function should have one parameter - a string
- Should return the character that occurs most often in the string
- You can use the **count()** function

- `most_used_char('one two three')` should return `'e'`
- `most_used_char('mississippians')` should return `'s'`
- `most_used_char('radaration')` should return `'a'`
- `most_used_char('zzzzzzzzzzzzzzzzzzzzzzzzzz')` should return `'z'`
- `most_used_char('at one time you are is i ')` should return `' '`

```python
def most_used_char(string):
    most_count = 0
    most_letter = ''
    for c in string:
        if string.count(c) > most_count:
            most_count = string.count(c)
            most_letter = c
    return most_letter
```

implement swap_words

- The **`swap_words`** function should have three string parameter variables
  - **The first:** a sentence. **The second:** a word. **The third:** a word
- Should return a string that has the two words (second and third parameters) swapped

- `swap_words('the cold night was not warm', 'warm', 'cold')`
  - Should return: `'the warm night was not cold'`
- `swap_words('the first day after the second', 'second', 'first')`
  - Should return: `'the second day after the first'`

```python
def swap_words(words, first, second):
    words = words.split()
    i = 0
    while i < len(words):
        if words[i] == first:
            words[i] = second
        if words[i] == second:
            words[i] = first
        i += 1
    result = ''
    for i in words:
        result += i + ' '
    return result
```

Why is this not correct?

Activity

```
def swap_words(words, first, second):
    words = words.split()
    i = 0
    while i < len(words):
        if words[i] == first:
            words[i] = second
        if words[i] == second:
            words[i] = first
        i += 1
    result = ''
    for i in words:
        result += i + ' '
    return result
```

```python
def swap_words(words, first, second):
    words = words.split()
    i = 0
    while i < len(words):
        if words[i] == first:
            words[i] = second
        elif words[i] == second:
            words[i] = first
        i += 1
    result = ''
    for i in words:
        result += i + ' '
    return result
```

Activity

```
def swap_words(words, first, second):
    words = words.split()
    i = 0
    while i < len(words):
        if words[i] == first:
            words[i] = second
        elif words[i] == second:
            words[i] = first
        i += 1
    result = ''
    for i in words:
        result += i + ' '
    return result
```

What is wrong with this?

```
result = ''
for i in words:
    result += i + ' '
return result
```

What can we do to get rid of the space at the end?

```
result = ''
for i in words:
    result += i + ' '
result = result.strip(' ')
return result
```

What can we do to get rid of the space at the end?

```
return ' '.join(words)
```

What can we do to get rid of the space at the end?

```python
def swap_words(words, first, second):
    words = words.split()
    i = 0
    while i < len(words):
        if words[i] == first:
            words[i] = second
        elif words[i] == second:
            words[i] = first
        i += 1
    return ' '.join(words)
```

```python
def swap_words(words, first, second):
    words = words.split()
    index_1 = words.index(first)
    index_2 = words.index(second)
    words[index_1], words[index_2] = words[index_2], words[index_1]
    return ' '.join(words)
```

Day strings

- We can represent days as strings in computer programs

```
first_day_of_class = '8/20/2018'
exam_1_day = '9/21/2018'
exam_2_day = '10/19/2018'
```

Day strings

- We can represent days as strings in computer programs

```
first_day_of_class = '8/20/2018'
exam_1_day = '9/21/2018'
exam_2_day = '10/19/2018'
```

However . . . .

What will it print?

```
first_day_of_class = '8/20/2018'
exam_1_day = '9/21/2018'
exam_2_day = '10/19/2018'

a = first_day_of_class < exam_1_day
b = first_day_of_class < exam_2_day

print(a, b)
```

What will it print?

```
d_day = '6/6/1942'
independence_day = '6/4/1776'
german_reunification = '10/3/1990'

a = d_day > german_reunification
b = german_reunification < independence_day

print(a, b)
```

implement gt_day (greater_than_day)

- The **`gt_day`** function should take two strings as a parameter
  - Day strings of the form **`'M/D/Y'`**
- Should return **`True`** if the first parameter is a later day than the second one, **`False`** otherwise

- `gt_day('10/10/2010', '9/9/2000')` should return **`True`**
- `gt_day('10/10/2000', '9/9/2010')` should return **`False`**
- `gt_day('8/12/2018', '10/19/2018')` should return **`False`**
- `gt_day('5/5/2018', '5/5/2018')` should return **`False`**

Activity

```python
def gt_day(first, second):
    # ? ? ?
```

```python
print(gt_day('10/10/2010', '9/9/2000'))
print(gt_day('10/10/2000', '9/9/2010'))
print(gt_day('8/20/2018', '10/19/2018'))
print(gt_day('5/5/2018', '5/5/2019'))
```

Activity

```python
def gt_day(first, second):
    split_1 = first.split('/')
    split_2 = second.split('/')
    a_day   = int(split_1[1])
    a_month = int(split_1[0])
    a_year  = int(split_1[2])
    b_day   = int(split_2[1])
    b_month = int(split_2[0])
    b_year  = int(split_2[2])
    # ? ? ? ?


print(gt_day('10/10/2010', '9/9/2000'))
print(gt_day('10/10/2000', '9/9/2010'))
print(gt_day('8/20/2018', '10/19/2018'))
print(gt_day('5/5/2018', '5/5/2019'))
```

```python
def gt_day(first, second):
    split_1 = first.split('/')
    split_2 = second.split('/')
    a_day   = int(split_1[1])
    a_month = int(split_1[0])
    a_year  = int(split_1[2])
    b_day   = int(split_2[1])
    b_month = int(split_2[0])
    b_year  = int(split_2[2])

    if a_year != b_year:
        return a_year > b_year
    elif a_month != b_month:
        return a_month > b_month
    else:
        return a_day > b_day
```