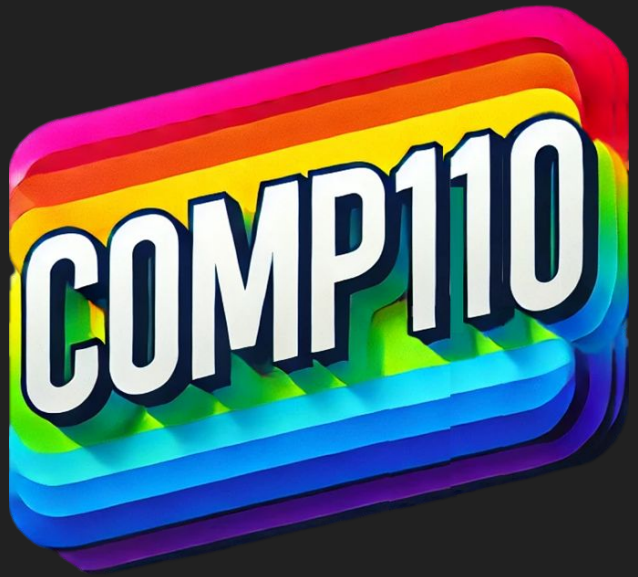

CL11 – Constants, Default Parameters, User Input, and Recursion Practice

Reminders

No class on Monday – Wellbeing Day!

- No Office Hours or Tutoring offered on Sunday or Monday (Feb 9 & 10)

Quiz 01 Prep:

- Review session in Fred Brooks (FB) 007 on Wednesday, Feb 12th at 6:15pm
- Practice quiz and solutions on the course site today
- Visit Office Hours and Tutoring – we *want* to help you!

Let’s play a guessing game!

Two new concepts:
named constants and **default parameters**

Named constants:

- Variables that are meant to hold a value that doesn't change throughout the program's execution
- Naming convention: all uppercase letters, with underscores between any words

```
PI: float = 3.14

FREEZING_F: int = 32

FREEZING_C: int = 0
```

Two new concepts:
named constants and **default parameters**

Default parameters:

- A parameter in a function signature that is assigned a default value. If a function call does not provide a value for that parameter, the default value is used
- *** Default parameters should come ***after*** non-default parameters in the function signature ***

```
def greet(name: str, greeting: str ="Hello") -> None:
```

default parameter

Two new concepts:
named constants and **default parameters**

Default parameters:

- A parameter in a function signature that is assigned a default value. If a function call does not provide a value for that parameter, the default value is used
- *** Default parameters should come ***after*** non-default parameters in the function signature ***

```
def greet(name: str, greeting: str ="Hello") -> None:
```

Happy with the default greeting? `greet(name="Conor")`

Want to specify your own greeting? `greet(name="Conor", greeting="Hi")`

Let’s write this function together, step by step!

Step 1: strings of different lengths?

word:

`len(word)`

!=

secret:

`len(secret)`

If the strings are different lengths, we automatically know they aren't the same string!

Step 2: determine if word's and secret's characters at a given index are different

word: secret:

bark `word[idx]` **!=** **punk** `punk[idx]`

If these characters are NOT equal, we automatically know **`word`**'s and **`secret`**'s values are different! No need to test the rest of the characters; just return **`False`**

Step 2: determine if word's and secret's characters at a given index are different

word: secret:

punt == **punk**

`word[idx]` `punk[idx]`

If these characters ARE the same (equal), we want to check of the characters at the next index (if there is another index to check) are equal!

Step 2: determine if word's and secret's characters at a given index are different

word: secret:

If these characters ARE the same (equal), we want to check of the characters at the next index (if there is another index to check) are equal!
Let's brainstorm how to write that in code…

Step 3: if we compared the characters at every index of **`word`** and **`secret`**, and they were always the same, we can return **`True`**

word: secret:

punk **punk**

`word[idx]` == `punk[idx]`

We guessed the secret word correctly!

```python
SECRET: str = "punk"

def guess_secret(word: str, secret: str, idx: int = 0) -> bool:
    # TODO 1
    if len(word) != len(secret):  # Different lengths? Not the same word!
        print("Words are different lengths.")
        return False
    # TODO 2
    if idx < len(word):  # As long as there are more characters to check...
        if word[idx] != secret[idx]:  # Characters at index aren't equal
            print(f"{word[idx]} isn't the secret word's next letter.")
            return False
        else:  # If the characters at that index WERE equal...
            print(
                f"{word[idx]} is at index {idx} for both words! Checking next letters..."
            )
            return guess_secret(word=word, secret=secret, idx=idx + 1)
    # TODO 3
    print("They are the same word!")
    return True

print(guess_secret(word=input("What is your word? "), secret=SECRET))
```