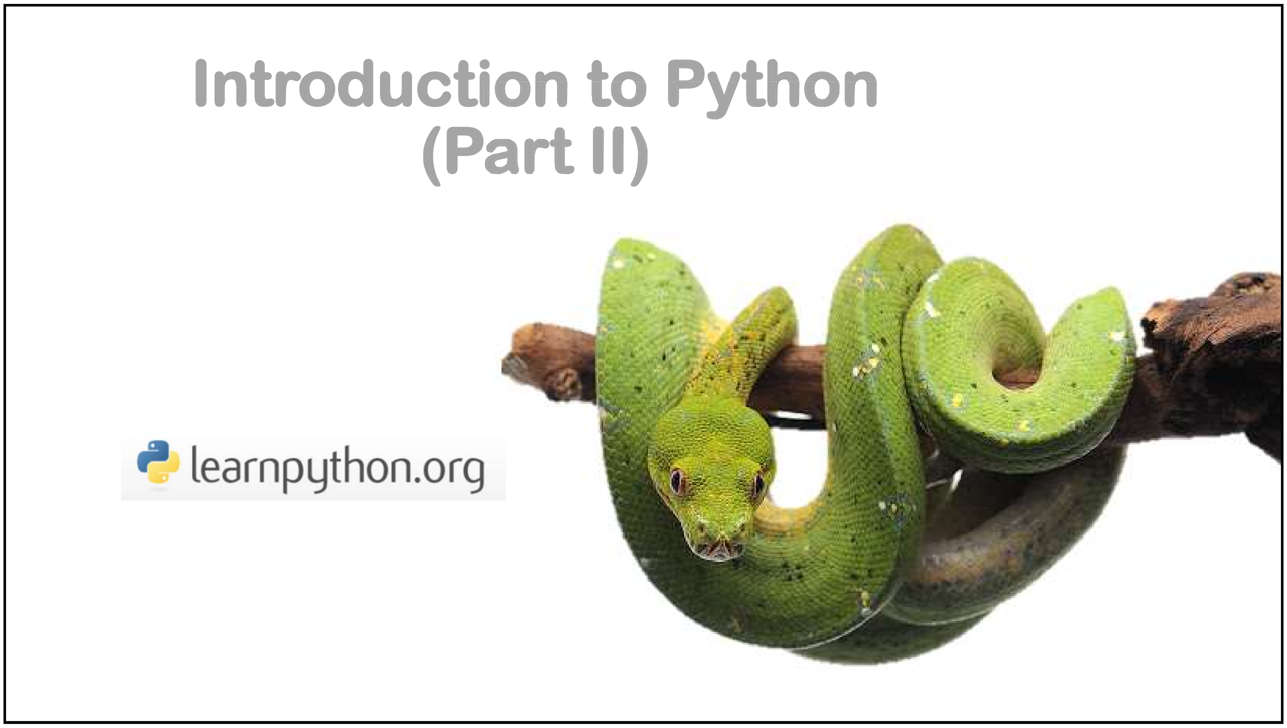
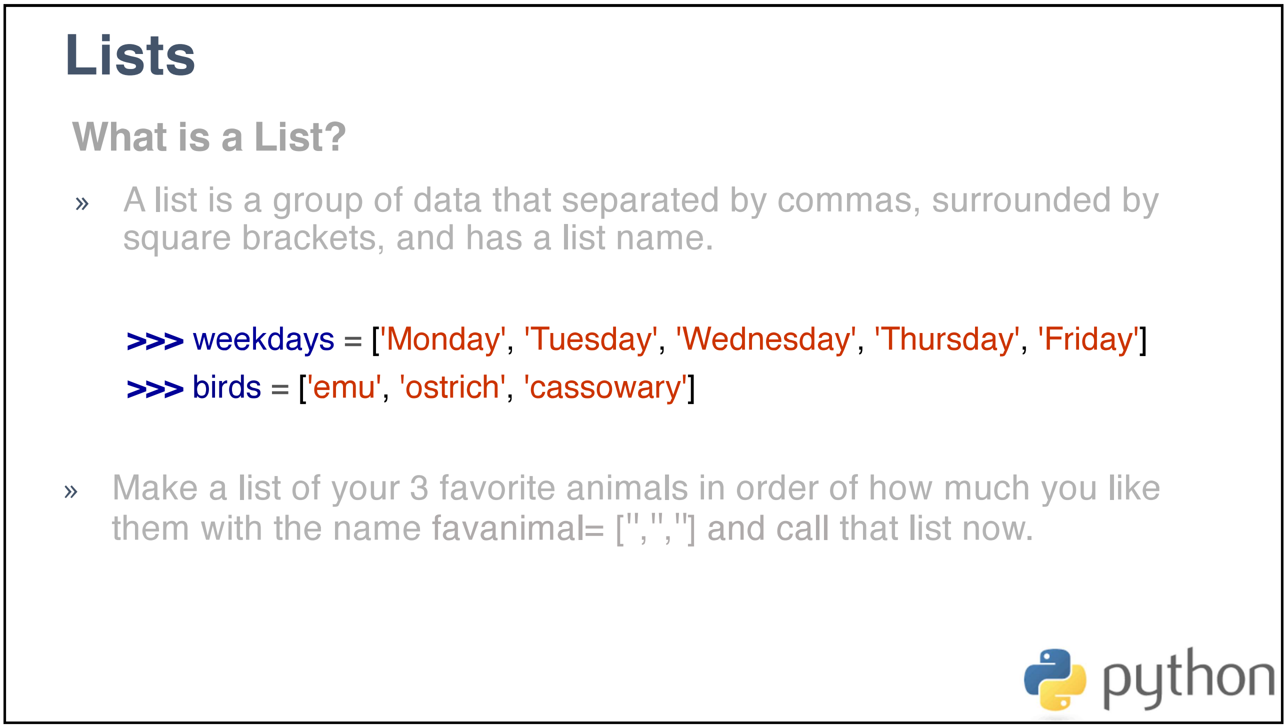


Introduction to Python (Part II)

learnpython.org

1



Lists

What is a List?

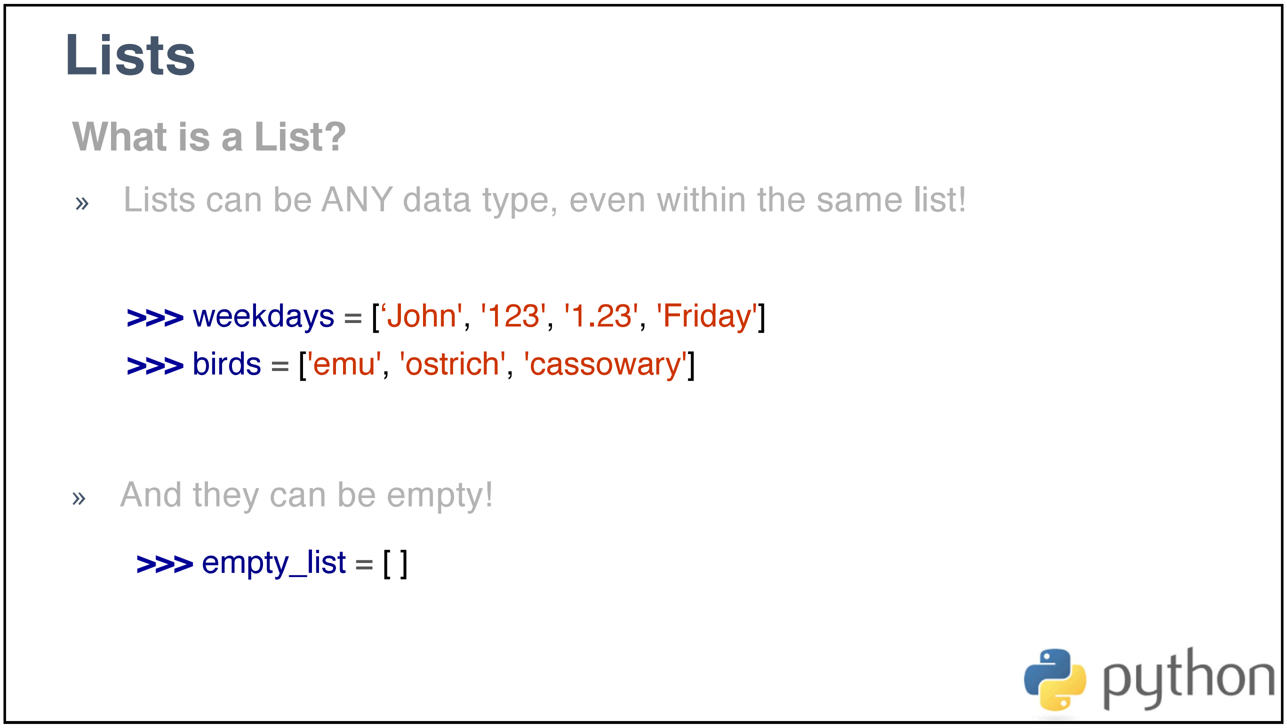
» A list is a group of data that separated by commas, surrounded by square brackets, and has a list name.

```
>>> weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
>>> birds = ['emu', 'ostrich', 'cassoway']
```

» Make a list of your 3 favorite animals in order of how much you like them with the name favanimal= ['','',''] and call that list now.

python

2



Lists

What is a List?

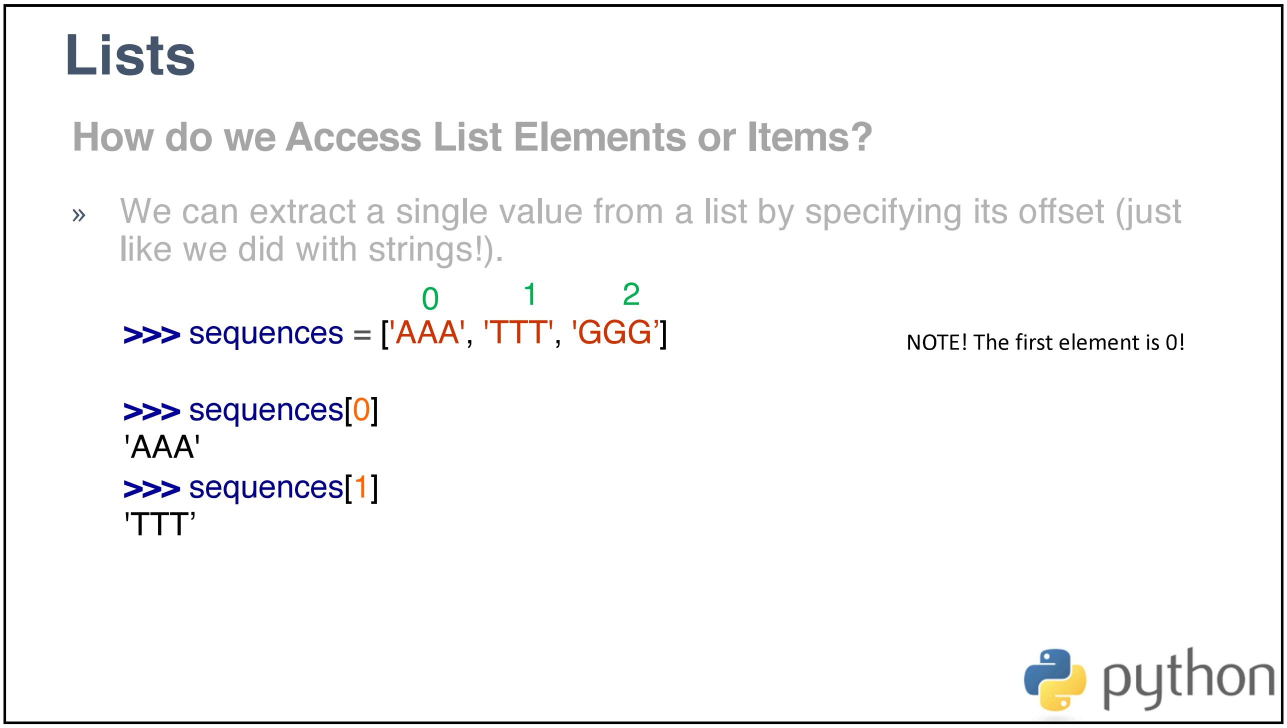
» Lists can be ANY data type, even within the same list!

```
>>> weekdays = [‘John', '123', '1.23', 'Friday']
>>> birds = ['emu', 'ostrich', 'cassowary']
```

» And they can be empty!

```
>>> empty_list = [ ]
```

3



Lists

How do we Access List Elements or Items?

» We can extract a single value from a list by specifying its offset (just like we did with strings!).

```
                  0      1      2
>>> sequences = ['AAA', 'TTT', 'GGG’]

>>> sequences[0]
'AAA'
>>> sequences[1]
'TTT’
```

NOTE! The first element is 0!

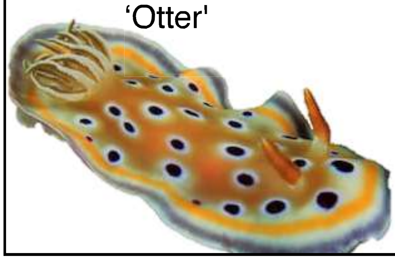
4

Lists

How do we Access List Elements or Items?

» What is your second favourite animal in your list?

```
                 0      1        2
>>> favanimal = ['Fox', ‘Otter', 'Nudibranch']

>>> sequences[1]
‘Otter'
```



5

Lists

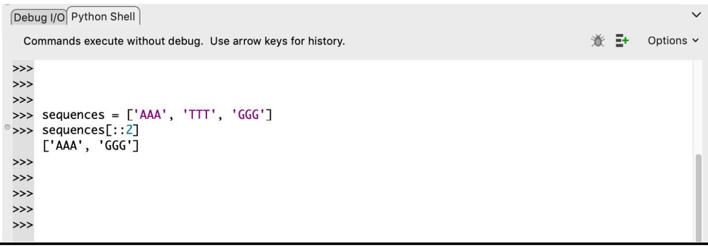
Get a Slice to Extract Items by Offset Range

» Extract a subsequence of a list by using a *slice*

```
>>> sequences = ['AAA', 'TTT', 'GGG']
>>> sequences[0:2]   # get items from position 0 and 1
['AAA', 'TTT']
```

» As with strings, slices can step by values other than one.

```
>>> sequences[::2] # get the first element and jump by two
['AAA', 'GGG']
```



Commands execute without debug. Use arrow keys for history.

```
>>>
>>>
>>>
>>> sequences = ['AAA', 'TTT', 'GGG']
>>> sequences[::2]
    ['AAA', 'GGG']
>>>
>>>
>>>
>>>
>>>
```

6

A little bit about functions, You will learn much more later in the course:

- A Function is a block of code.
- You can run functions by calling them
- In python functions look like this
  Function_name()
- Pay attention to the parenthesis!

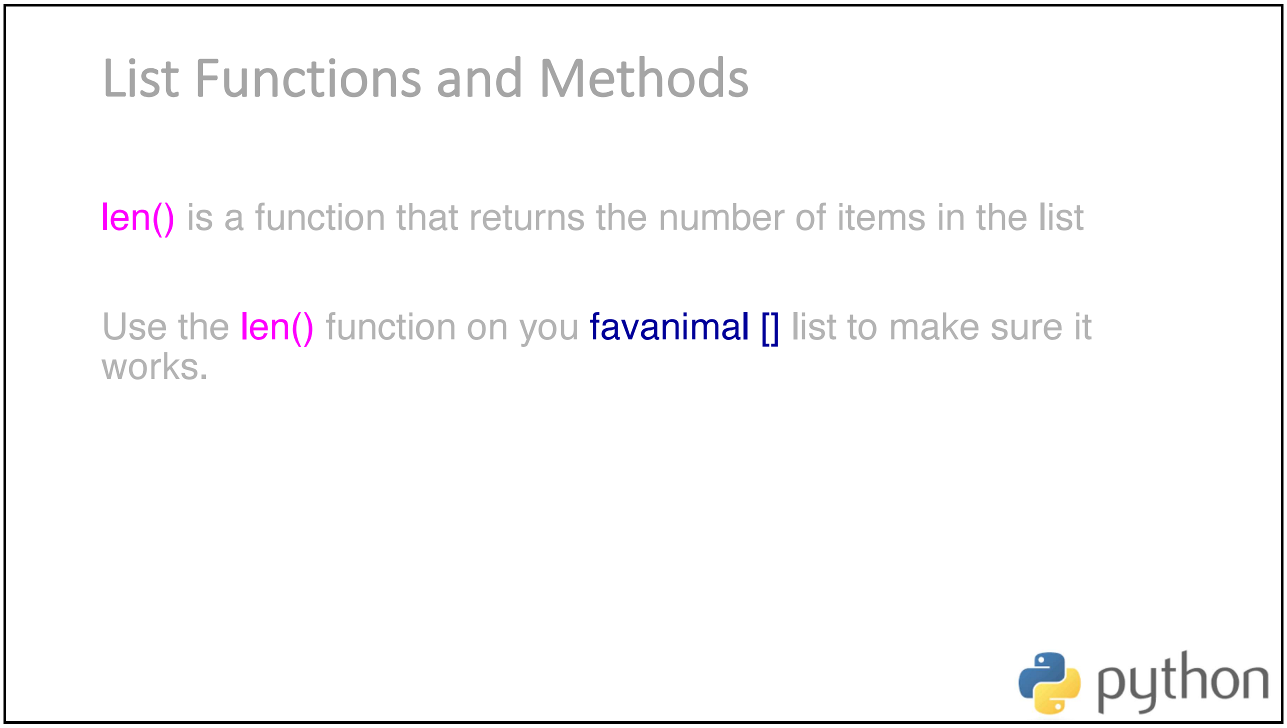
7

List Functions and Methods

len() is a function that returns the number of items in the list

```
>>> sequences = ['AAA', 'TTT', 'GGG’, ‘CCC’]
>>> len(sequences)
4
```

8

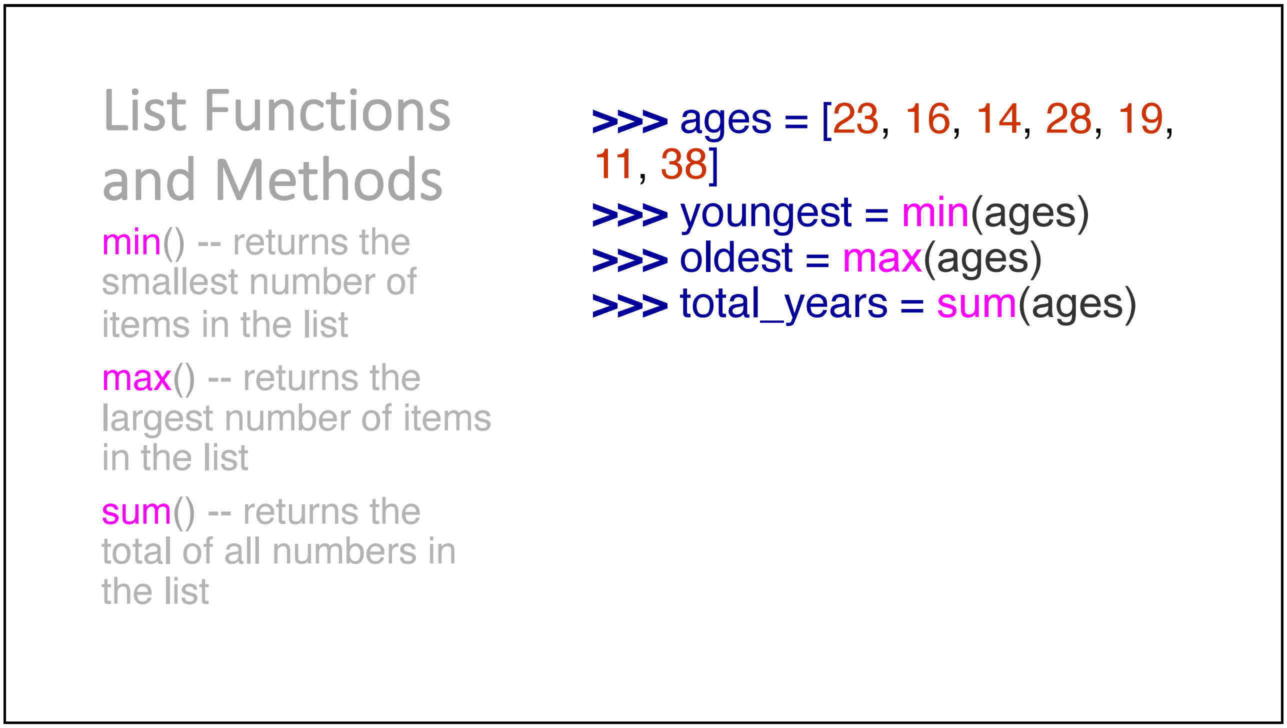


List Functions and Methods

len() is a function that returns the number of items in the list

Use the len() function on you favanimal [] list to make sure it works.

9



List Functions and Methods

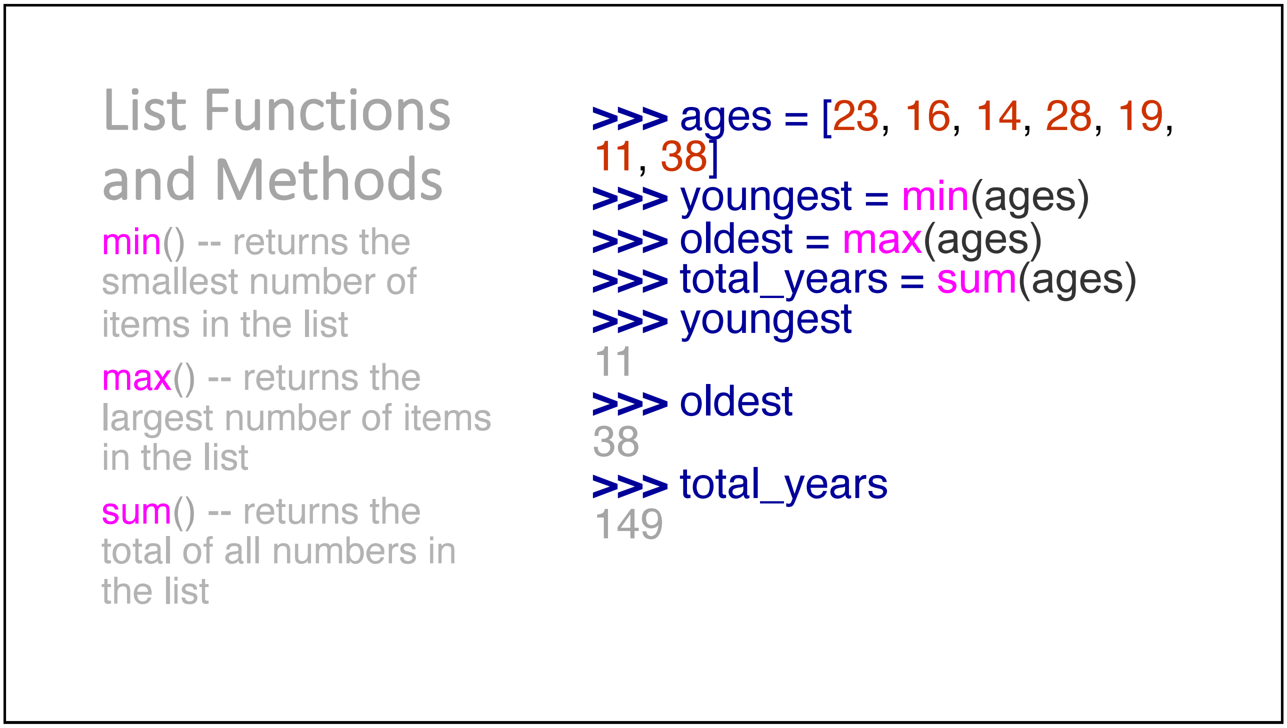
min() -- returns the smallest number of items in the list

max() -- returns the largest number of items in the list

sum() -- returns the total of all numbers in the list

```
>>> ages = [23, 16, 14, 28, 19,
11, 38]
>>> youngest = min(ages)
>>> oldest = max(ages)
>>> total_years = sum(ages)
```

10



List Functions and Methods

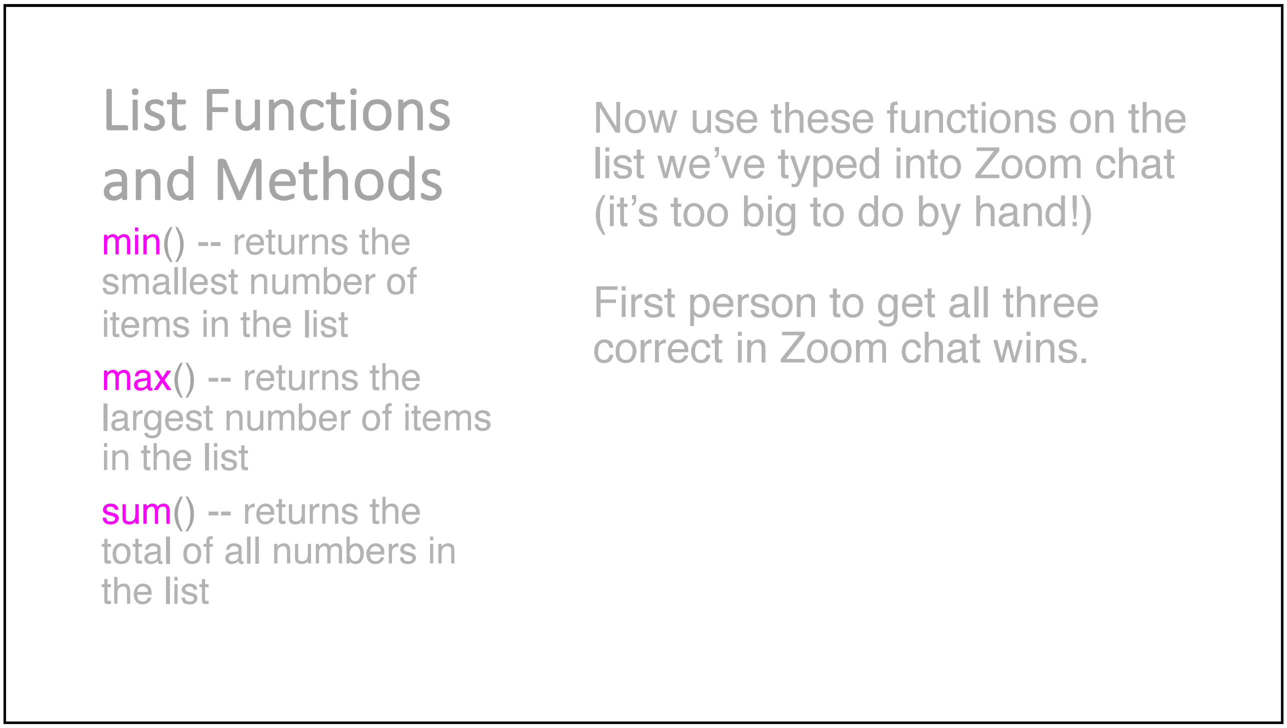
min() -- returns the smallest number of items in the list

max() -- returns the largest number of items in the list

sum() -- returns the total of all numbers in the list

```
>>> ages = [23, 16, 14, 28, 19,
11, 38]
>>> youngest = min(ages)
>>> oldest = max(ages)
>>> total_years = sum(ages)
>>> youngest
11
>>> oldest
38
>>> total_years
149
```

11



List Functions and Methods

min() -- returns the smallest number of items in the list

max() -- returns the largest number of items in the list

sum() -- returns the total of all numbers in the list

Now use these functions on the list we’ve typed into Zoom chat (it’s too big to do by hand!)

First person to get all three correct in Zoom chat wins.

12

List Functions and Methods

- list.sort() -- sort the list itself, in place.

Question for the students: What is the difference between list.sort() and sort()?

```
>>> sequences = ['AAA', 'TTT', 'GGG']
>>> sequences.sort()
>>> sequences
['AAA', 'GGG', 'TTT']
>>> numbers = [2, 1, 4.0, 3]
>>> numbers.sort()
>>> numbers
[1, 2, 3, 4.0]
```

13

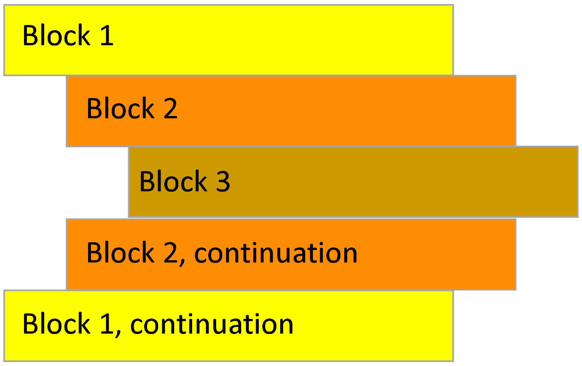
Control Statements

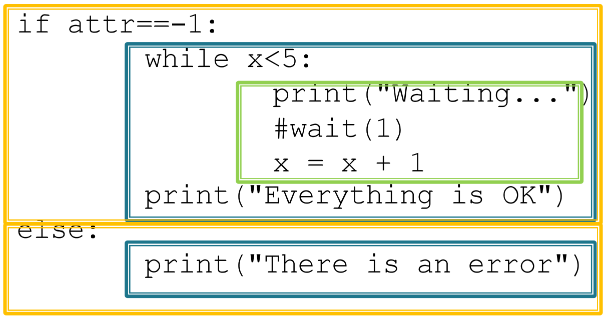
Do not mix space and tab together!!!

Indentation

» Code blocks are defined by their indentation.

» Leading whitespace (spaces and tabs) at the beginning of a logical line is used to compute the indentation level of the line.



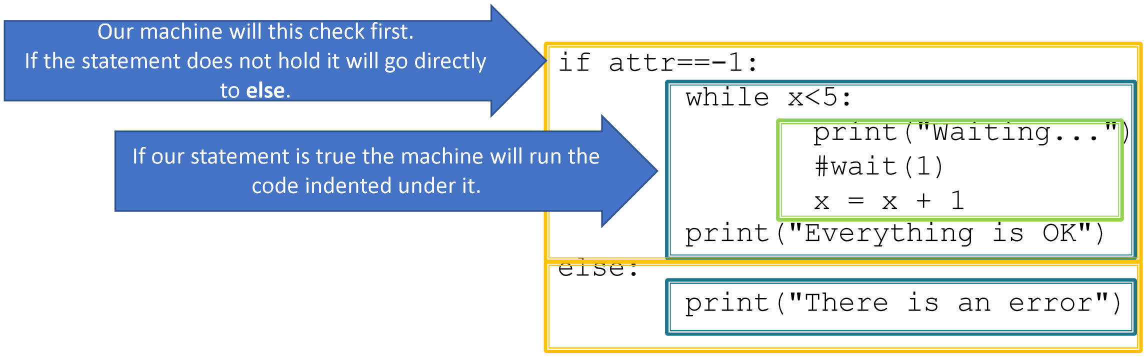


```
if attr==-1:
       while x<5:
              print("Waiting...")
              #wait(1)
              x = x + 1
       print("Everything is OK")
else:
       print("There is an error")
```

14

Indentation

Indentation gives priority to a line of code, lets look at our example



Our machine will this check first.
If the statement does not hold it will go directly to **else**.

If our statement is true the machine will run the code indented under it.

```
if attr==-1:
      while x<5:
            print("Waiting...")
            #wait(1)
            x = x + 1
      print("Everything is OK")
else:
      print("There is an error")
```

15

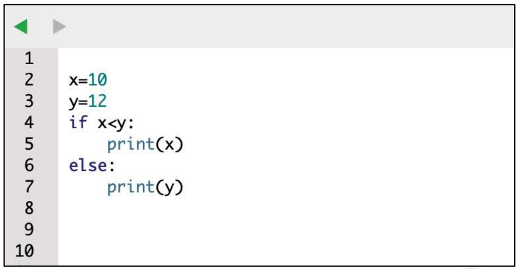
Control Statements

Conditional Statements: IF-ELSE

» If statement evaluates an expression. If the expression is true, the block of code just after the if clause is executed. Otherwise, the block under else is executed.

```
if EXPRESSION1:
       STATEMENT1
elif EXPRESSION2:
       STATEMENT2
elif EXPRESSION3:
       STATEMENT3
else:
       STATEMENT4
```

elif is optional, you can use as many elif's as you want.



```

x=10
y=12
if x<y:
    print(x)
else:
    print(y)
```

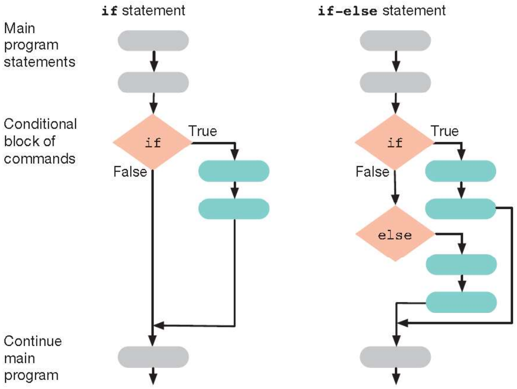
16

Control Statements

Conditional Statements: IF-ELSE

» Take into account that once a condition is evaluated as true, the remaining conditions are not checked.

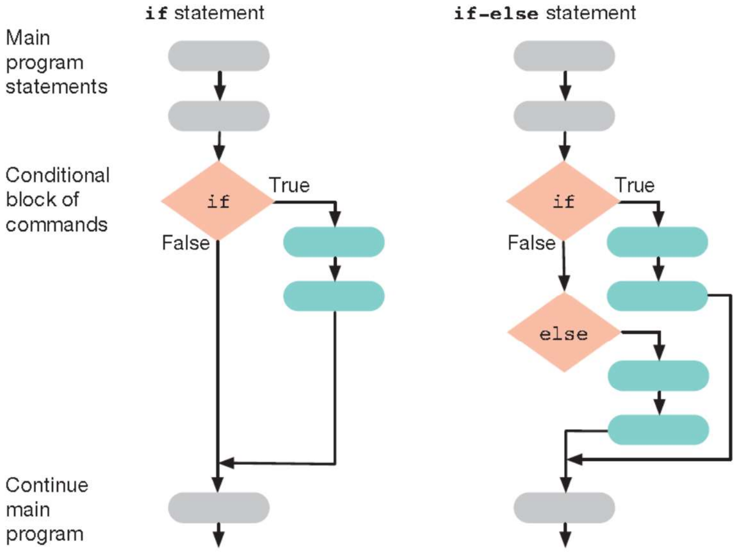
```
if EXPRESSION1:              if EXPRESSION1:
        STATEMENT1                   STATEMENT1
elif EXPRESSION2:     VS.    if EXPRESSION2:
        STATEMENT2                   STATEMENT2
elif EXPRESSION3:            if EXPRESSION3:
        STATEMENT3                   STATEMENT3
else:                        if EXPRESSION4:
        STATEMENT4                   STATEMENT4
```



What's the difference between these two?

17

Fill in the bubbles with our Example

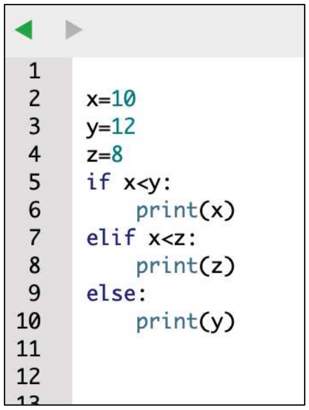


18

Control Statements

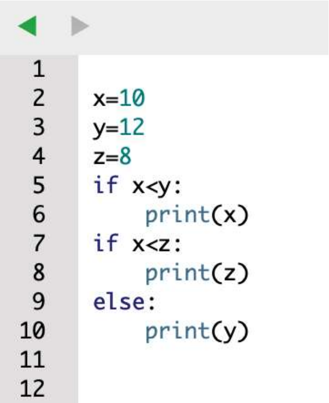
Conditional Statements: IF-ELSE

» Take into account that once a condition is evaluated as true, the remaining conditions are not checked.



```
x=10
y=12
z=8
if x<y:
    print(x)
elif x<z:
    print(z)
else:
    print(y)
```

VS.



```
x=10
y=12
z=8
if x<y:
    print(x)
if x<z:
    print(z)
else:
    print(y)
```

10

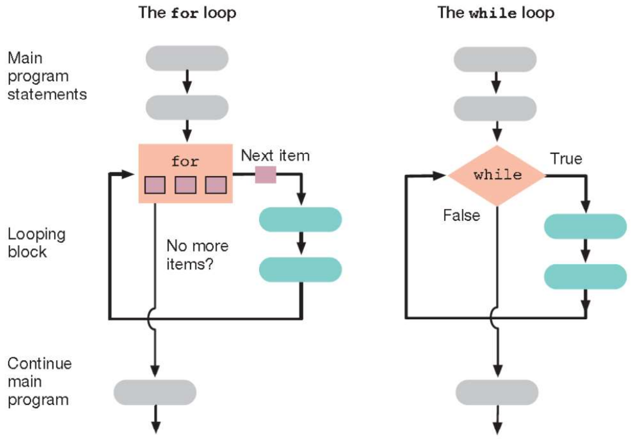
10
12

19

Control Statements

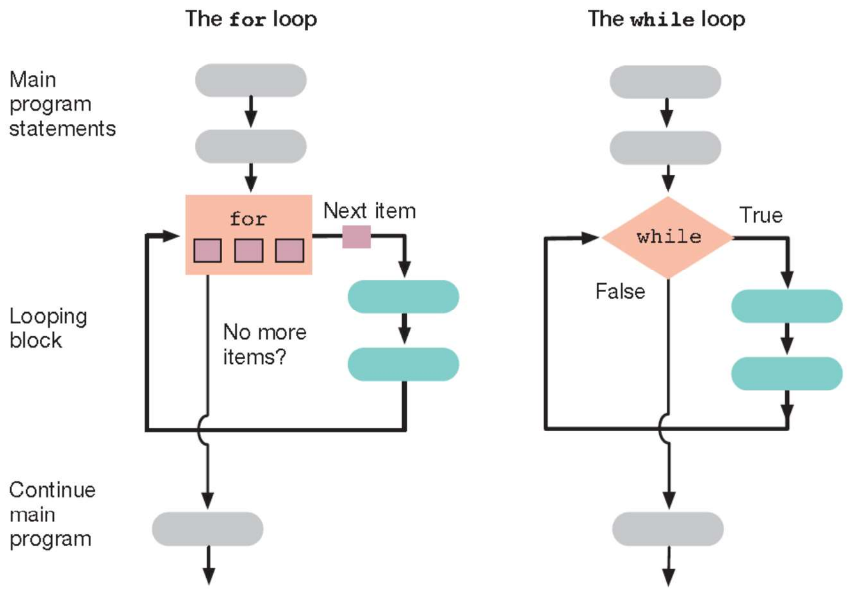
Loops: for and while loops

» A loop is a block of statements that gets executed repeatedly as long as some condition is true.



20

Fill in the bubbles with our Example



Control Statements

Loops: for loops

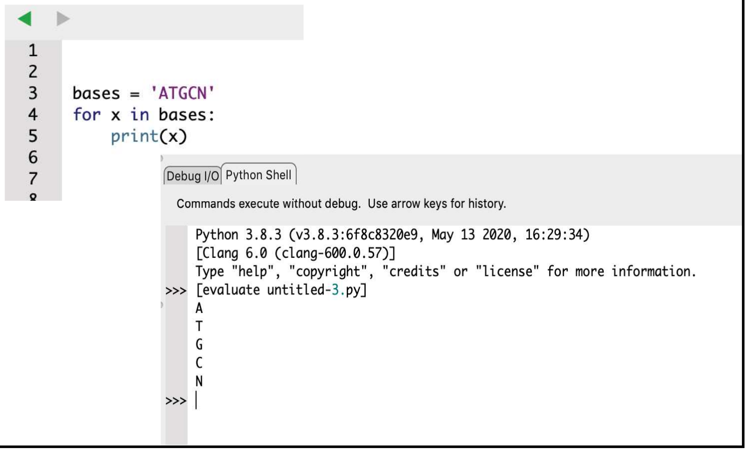
- » A for loop structure allows code to be repeatedly executed while keeping a variable with the value of an iterable object.
- » Iterable objects: lists, tuples, strings and dictionaries.
- » A for loop structure:

```
for VAR in ITERABLE:
        Statement
```

```
bases = 'ATGCN'
for x in bases:
    print(x)
```

→

```
A
T
G
C
N
```



```


bases = 'ATGCN'
for x in bases:
    print(x)
```

Commands execute without debug. Use arrow keys for history.

```
    Python 3.8.3 (v3.8.3:6f8c8320e9, May 13 2020, 16:29:34)
    [Clang 6.0 (clang-600.0.57)]
    Type "help", "copyright", "credits" or "license" for more information.
>>> [evaluate untitled-3.py]
    A
    T
    G
    C
    N
>>> |
```

Control Statements

Loops: while loops

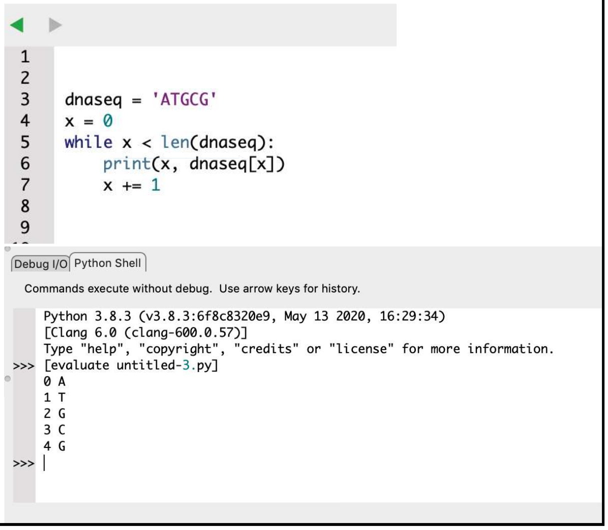
» while loop executes a set of statements as long as a condition is true.

» while loop structure

```
while EXPRESSION:
        STATEMENT
```

```
dnaseq = 'ATGCG'
x = 0
while x < len(dnaseq):
    print(x, dnaseq[x])
    x += 1
```

```
0 A
1 T
2 G
3 C
4 G
```



```
dnaseq = 'ATGCG'
x = 0
while x < len(dnaseq):
    print(x, dnaseq[x])
    x += 1
```

Commands execute without debug. Use arrow keys for history.

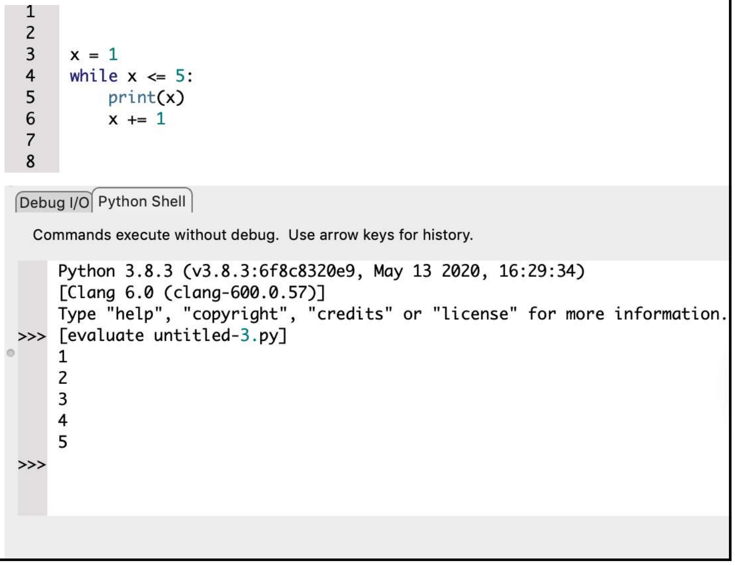
```
    Python 3.8.3 (v3.8.3:6f8c8320e9, May 13 2020, 16:29:34)
    [Clang 6.0 (clang-600.0.57)]
    Type "help", "copyright", "credits" or "license" for more information.
>>> [evaluate untitled-3.py]
    0 A
    1 T
    2 G
    3 C
    4 G
>>> |
```

Control Statements

Loops: while loops

```
x = 1
while x <= 5:
    print(x)
    x += 1
```

```
1
2
3
4
5
```



```
x = 1
while x <= 5:
    print(x)
    x += 1
```

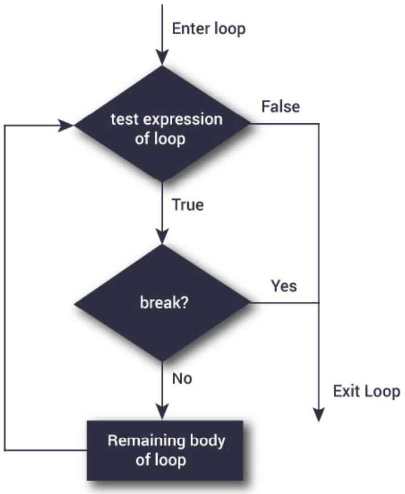
Commands execute without debug. Use arrow keys for history.

```
    Python 3.8.3 (v3.8.3:6f8c8320e9, May 13 2020, 16:29:34)
    [Clang 6.0 (clang-600.0.57)]
    Type "help", "copyright", "credits" or "license" for more information.
>>> [evaluate untitled-3.py]
    1
    2
    3
    4
    5
>>>
```

Control Statements

Break Statement: break the loop

- break can be used to escape from a loop structure, either for or while loop



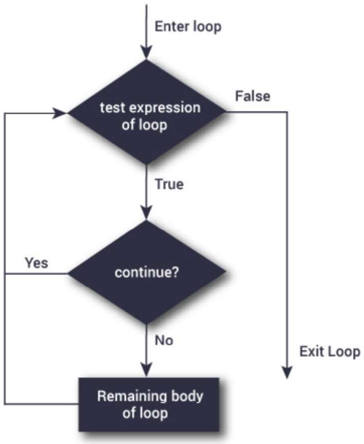
```
for var in sequence:
    # codes inside for loop
    if  condition:
        break
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if  condition:
        break
    # codes inside while loop

# codes outside while loop
```

Control Statements

Continue Statement

- The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.



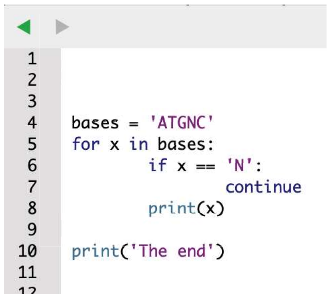
```
for var in sequence:
    # codes inside for loop
    if  condition:
        continue
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if  condition:
        continue
    # codes  inside while loop

# codes outside while loop
```

Control Statements

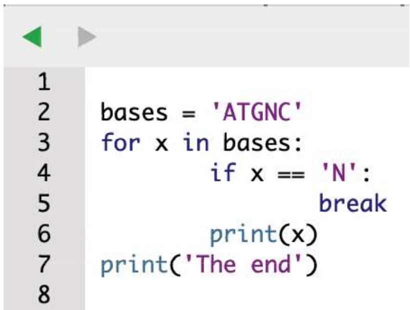
Continue Statement

- The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.



```
bases = 'ATGNC'
for x in bases:
        if x == 'N':
                continue
        print(x)

print('The end')
```

```
A
T
G
C
The end
```



```
bases = 'ATGNC'
for x in bases:
        if x == 'N':
                break
        print(x)
print('The end')
```

```
A
T
G
The end
```

Functions

Defining Functions of Your Own

- Code reuse or DRY (Don’t Repeat Yourself)
- A function is a piece of code written to carry out a specified task, and called by name.
- Two important things with a function
  - Define it
  - Call it
- Definition statements have the general form:

  def functionname(arg1, arg2, …, argN):
  statements
  return values

- Call the function ☎
  functionname(expression)

Functions

Defining Functions of Your Own

» Creating a function

function name

```
def Hello():
        print("Hello World!")
```

Define a function named "Hello".
Does not print anything until you call the function.

» Calling a function

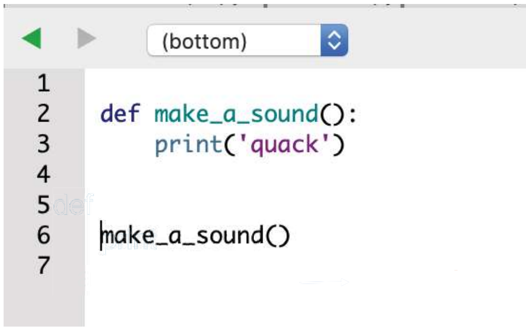
Hello() → Hello World!

» How function works

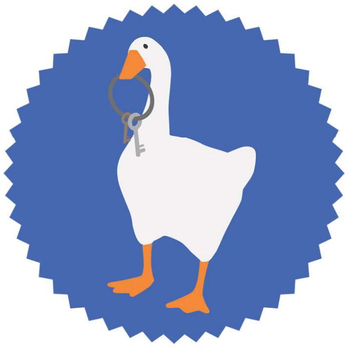
```
print("Hello User!")
print("Hello User!")
print("Hello User!")
print("Hello User!")
```

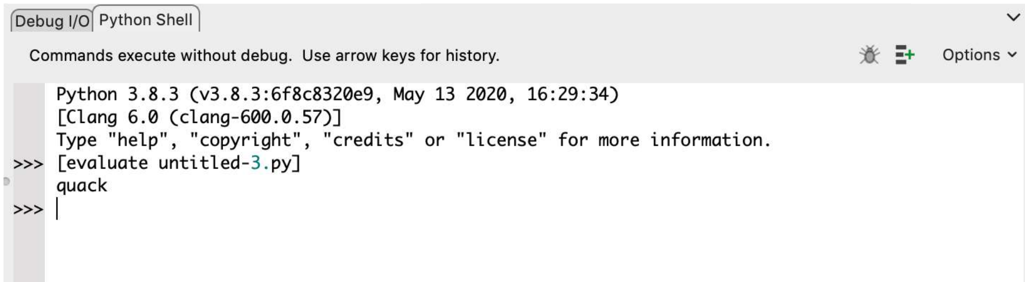
```
def Hello():
        print("Hello
User!")
Hello()
Hello()
Hello()
Hello()
```

29



```

def make_a_sound():
    print('quack')


make_a_sound()
```





Commands execute without debug. Use arrow keys for history.

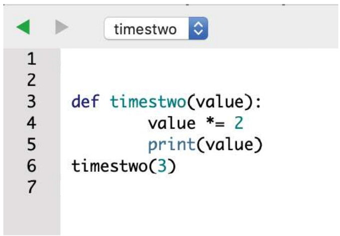
```
Python 3.8.3 (v3.8.3:6f8c8320e9, May 13 2020, 16:29:34)
[Clang 6.0 (clang-600.0.57)]
Type "help", "copyright", "credits" or "license" for more information.
>>> [evaluate untitled-3.py]
quack
>>> |
```

30

Functions

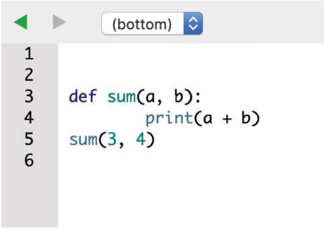
Defining Functions of Your Own

- Arguments and Parameters
  - Information can be passed to functions as Arguments
  - The values of those arguments are copied to their corresponding *parameters* inside the function
  - Assigns the first parameter name to the first argument's value, the second parameter name to the second value, and so on



```
def timestwo(value):
        value *= 2
        print(value)
timestwo(3)
```

```
6
```



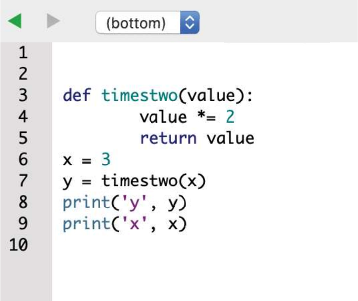
```
def sum(a, b):
        print(a + b)
sum(3, 4)
```

```
7
```

Functions

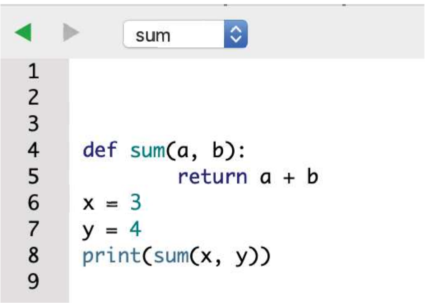
Defining Functions of Your Own

- Return value
  - The statement return [expression] exits a function, optionally passing back an expression to the caller.
  - A return statement with no arguments is the same as return None.



```
def timestwo(value):
        value *= 2
        return value
x = 3
y = timestwo(x)
print('y', y)
print('x', x)
```

```
y 6
x 3
```



```
def sum(a, b):
        return a + b
x = 3
y = 4
print(sum(x, y))
```

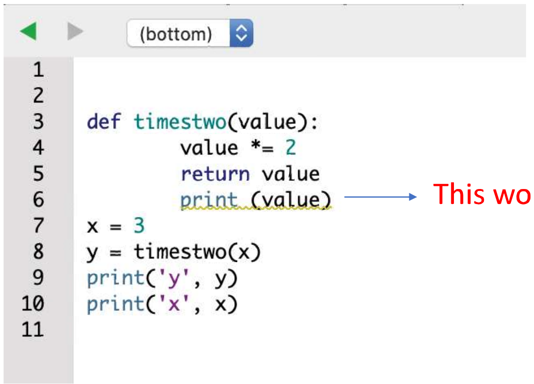
```
7
```

Functions

Defining Functions of Your Own

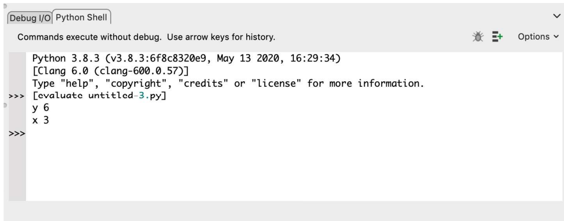
» Return value

- Once you return a value from a function, it immediately stops being executed. Any code after the **return** statement will never happen.



```
def timestwo(value):
        value *= 2
        return value
        print (value)
x = 3
y = timestwo(x)
print('y', y)
print('x', x)
```

This won't be printed!



Commands execute without debug. Use arrow keys for history.

```
    Python 3.8.3 (v3.8.3:6f8c8320e9, May 13 2020, 16:29:34)
    [Clang 6.0 (clang-600.0.57)]
    Type "help", "copyright", "credits" or "license" for more information.
>>> [evaluate untitled-3.py]
    y 6
    x 3
>>>
```

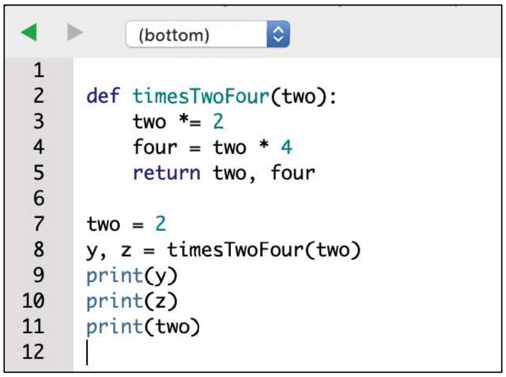
33

Functions

Defining Functions of Your Own

» Variable names inside methods are stored separately -- **Local Variables**

» **Global variables** defined and declared outside a function

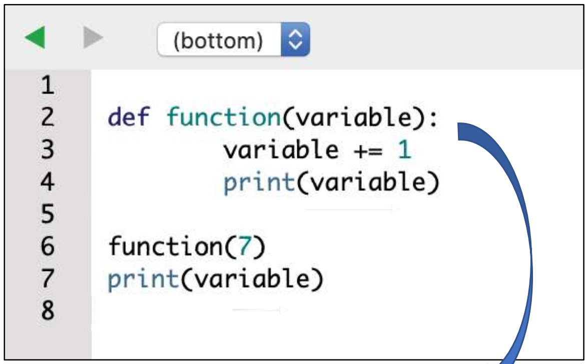


```
def timesTwoFour(two):
    two *= 2
    four = two * 4
    return two, four

two = 2
y, z = timesTwoFour(two)
print(y)
print(z)
print(two)
```

4

16

2



```
def function(variable):
        variable += 1
        print(variable)

function(7)
print(variable)
```

NameError: name ‘variable’ is not defined

34

Structured programming

Definition

- Modular programming, enforces a logical structure on the program being written to make it more efficient and easier to understand and modify.
- In SP, control of program flow is restricted to three structures, sequence, IF THEN ELSE, and DO WHILE, or to a structure derivable from a combination of the basic three.

Three main parts:

1. Loops (for and while)
2. Flow control (IF-ELSE)
3. Function

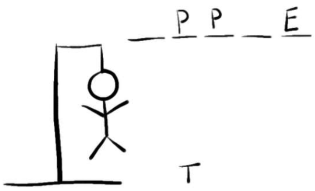
python

35

Structured programming

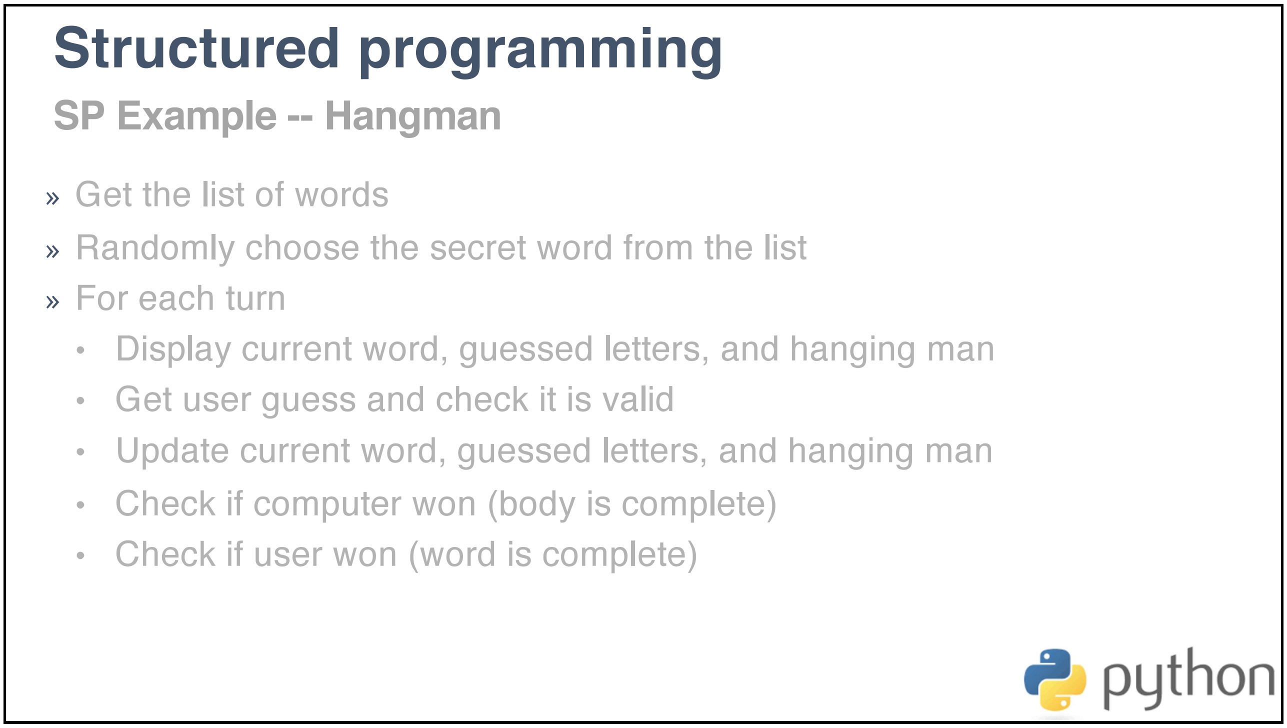
SP Example -- Hangman

- Hangman is a slightly macabre children's game
- Player one thinks of a secret word, indicated by a number of dashes
- On each turn, player two guesses one letter
  - if the letter occurs in the word, the blanks are filled in
  - if the letter does not occur, a "body part" is added to the man
- Player one wins if the body is completed before the word is guessed
- Player two wins is the word is guessed before the body is completed



python

36

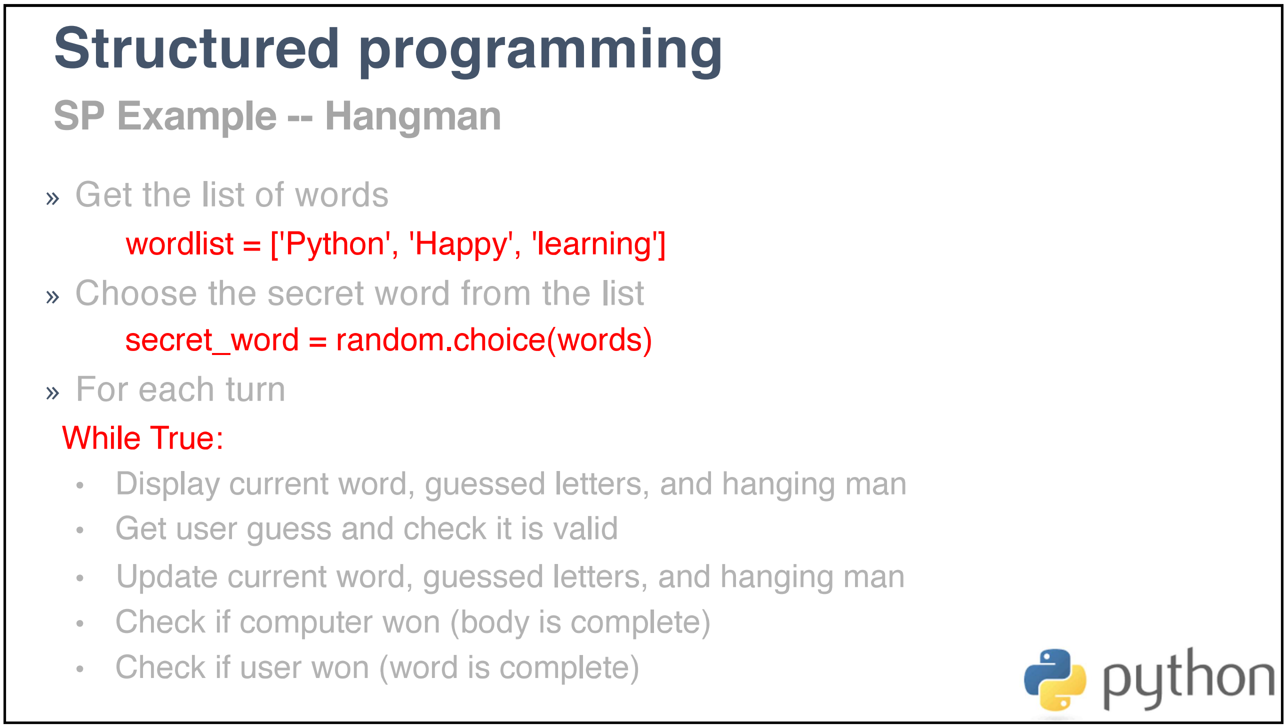


Structured programming

SP Example -- Hangman

- Get the list of words
- Randomly choose the secret word from the list
- For each turn
  - Display current word, guessed letters, and hanging man
  - Get user guess and check it is valid
  - Update current word, guessed letters, and hanging man
  - Check if computer won (body is complete)
  - Check if user won (word is complete)

37



Structured programming

SP Example -- Hangman

- Get the list of words

```
wordlist = ['Python', 'Happy', 'learning']
```

- Choose the secret word from the list

```
secret_word = random.choice(words)
```

- For each turn

```
While True:
```

  - Display current word, guessed letters, and hanging man
  - Get user guess and check it is valid
  - Update current word, guessed letters, and hanging man
  - Check if computer won (body is complete)
  - Check if user won (word is complete)

38

Structured programming

SP Example -- Hangman

» Get the list of words

wordlist = ['Python', 'Happy', 'learning']

» Choose the secret word from the list

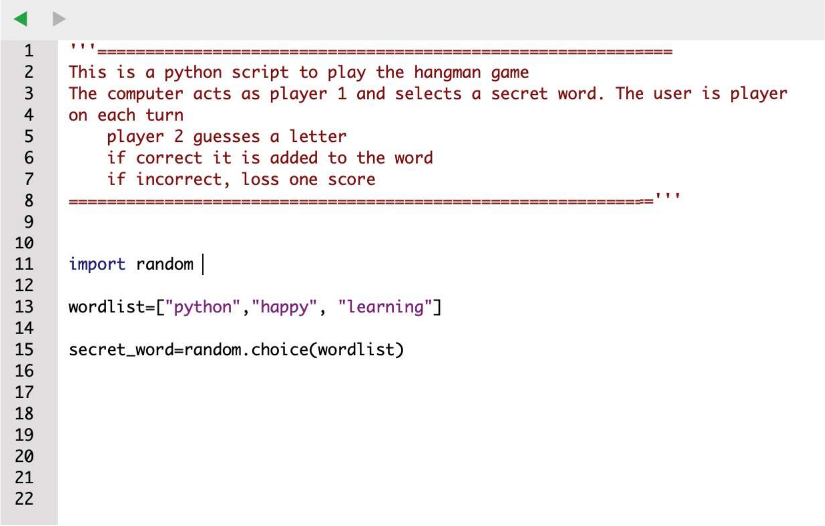
secret_word = random.choice(words)

```
'''============================================================
This is a python script to play the hangman game
The computer acts as player 1 and selects a secret word. The user is player
on each turn
    player 2 guesses a letter
    if correct it is added to the word
    if incorrect, loss one score
============================================================'''
import random

wordlist = ['Python', 'Happy', 'learning']

secret_word = random.choice(wordlist)
```

39

Structured programming

SP Example -- Hangman



```
'''===========================================================
This is a python script to play the hangman game
The computer acts as player 1 and selects a secret word. The user is player
on each turn
     player 2 guesses a letter
     if correct it is added to the word
     if incorrect, loss one score
============================================================'''


import random 

wordlist=["python","happy", "learning"]

secret_word=random.choice(wordlist)
```

40

Structured programming

SP Example -- Hangman

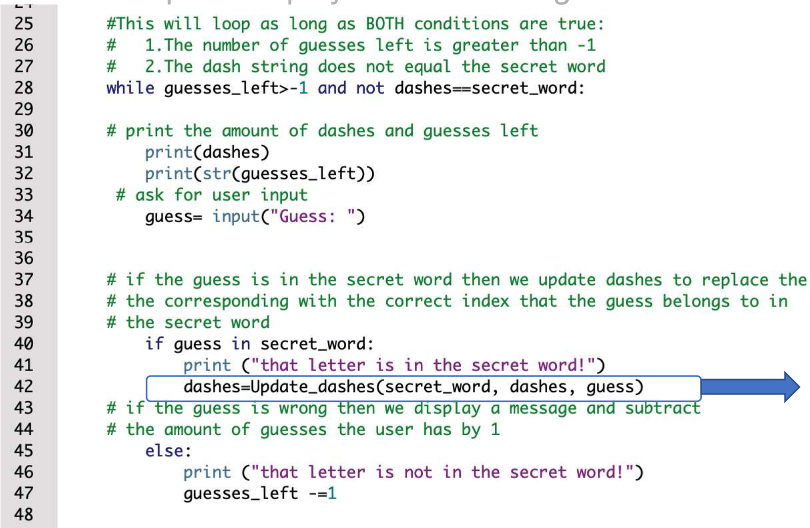
- Define GetGuess function

```
def GetGuess():
    '''Set the dashes to the length of the secret word and set the amount of guesses
     let the player guess if the letter in the selected word
     '''
    dashes = "-" * len(secret_word)
    guesses_left = 10
```

Structured programming

SP Example -- Hangman

- Define GetGuess function
  - Use loops to let player to enter the guessed letter



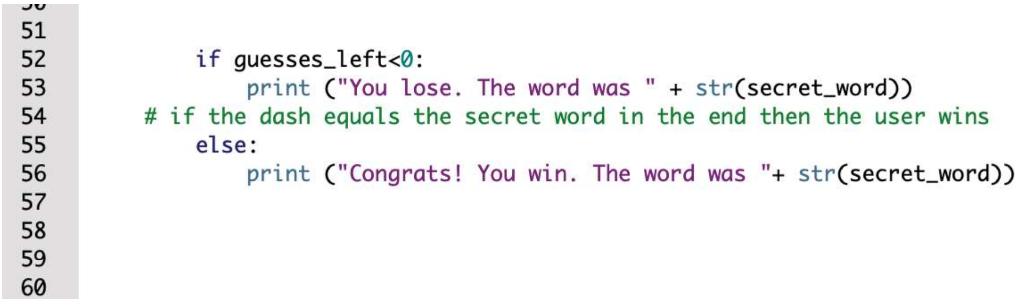
```
#This will loop as long as BOTH conditions are true:
#   1.The number of guesses left is greater than -1
#   2.The dash string does not equal the secret word
while guesses_left>-1 and not dashes==secret_word:

# print the amount of dashes and guesses left
    print(dashes)
    print(str(guesses_left))
 # ask for user input
    guess= input("Guess: ")


# if the guess is in the secret word then we update dashes to replace the
# the corresponding with the correct index that the guess belongs to in
# the secret word
    if guess in secret_word:
        print ("that letter is in the secret word!")
        dashes=Update_dashes(secret_word, dashes, guess)
# if the guess is wrong then we display a message and subtract
# the amount of guesses the user has by 1
    else:
        print ("that letter is not in the secret word!")
        guesses_left -=1
```

Call Function
update_dashes

Structured programming

SP Example -- Hangman

- Define GetGuess function
  - After the while loop is False



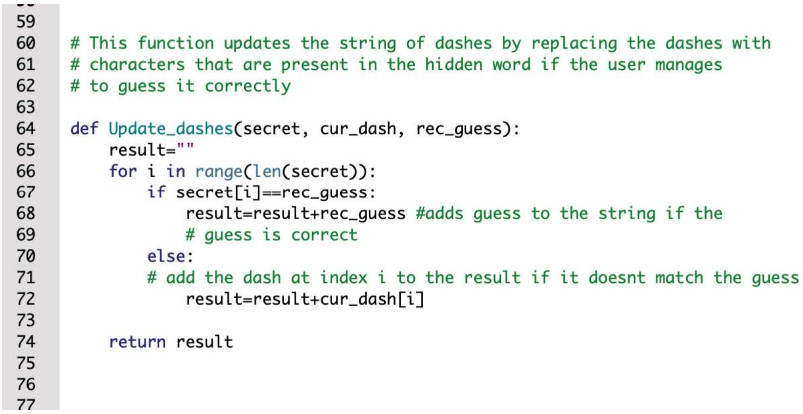
```
        if guesses_left<0:
            print ("You lose. The word was " + str(secret_word))
    # if the dash equals the secret word in the end then the user wins
        else:
            print ("Congrats! You win. The word was "+ str(secret_word))
```

Structured programming

SP Example -- Hangman

- Second function – Update dashes

dashes = update_dashes(secret_word, dashes, guess)



```
# This function updates the string of dashes by replacing the dashes with
# characters that are present in the hidden word if the user manages
# to guess it correctly

def Update_dashes(secret, cur_dash, rec_guess):
    result=""
    for i in range(len(secret)):
        if secret[i]==rec_guess:
            result=result+rec_guess #adds guess to the string if the
            # guess is correct
        else:
        # add the dash at index i to the result if it doesnt match the guess
            result=result+cur_dash[i]

    return result
```

Structured programming

SP Example -- Hangman

» Call GetGuess Function

GetGuess()

» **In Class Exercises**

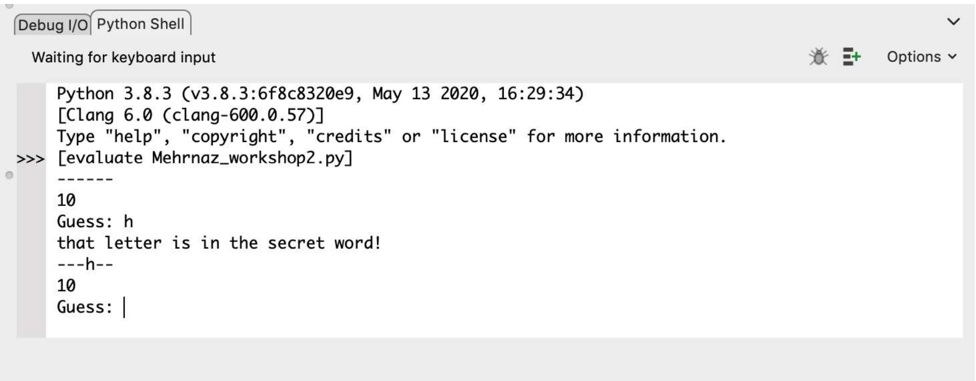
» To use the 3 uploaded part scripts to combine them together into a complete script to let it work!!! (Don't worry, it will take some fiddling with!)

45

Don't forget to CALL!

- Call your function to run it:

  GetGuess()



```
Python 3.8.3 (v3.8.3:6f8c8320e9, May 13 2020, 16:29:34)
[Clang 6.0 (clang-600.0.57)]
Type "help", "copyright", "credits" or "license" for more information.
>>> [evaluate Mehrnaz_workshop2.py]
------
10
Guess: h
that letter is in the secret word!
---h--
10
Guess: 
```

46

Notes

Notes