Strings Functions

Introduction to Computer Science!

Freq AB

Python Variables

```
>> x = 41
>> y = x + 1
>> x = x + y
>> y
?? (1)
>> x
??
```

41 + 42

What value is displayed for y at ??(1)?

A. 41

B. 42

C. 83

D. 84

x ~~41~~ 83

y 42

x

y

Strings

- A string is a sequence of characters
- Anything within single or double quotes: E.g "UCSB", '73$505abc' "UC\"SB" , 'UCSB'

  numbers

  other characters
- What about an empty string? "" or ''
- What if a string includes a quote e.g. UC"SB?

  >>> x = "UC\"SB" x = "UC\"SB"

  print(x)

  UC"SB
- How can we get the length of the string?

  >>> len("UCSB")

  4

String operations

- Concatenation: +
- Repetition: * → creating copies
- Parse and extract → Getting a substring from a string or string
- Check if some character is in a string
- Compare == != < >

"apple" < "bat" → True

True or False

Lexicographical ordering is used

"ap" in "apple" → True

"ca" in "apple" → False

"ca" not in "apple" → True

String operations: Concatenate

- Concatenation: str + str
- What is “Hello” + “ World”?

"Hello World"

String operations: Repetition

- Repeat: str * int
- What is “Hello” *3?

"HelloHelloHello"

What is the value of s after the following code runs?

```
s = 'abc'
s = 'd' * 3 + s
s = s + e* 2
```

s = 'abc'

→ s = 'd' * 3 + s

s = s + 'e' * 2

c

s = 'ddd' + 'abc'

A. 'abcd3e2'

B. 'abcdddabc'

C. 'dddabcee'

D. 'abcdddabce2'

E. Error Variable e was never defined, so it cannot be used in an expression.

Parsing strings…

Parsing means extracting certain pieces from a string

schoolName = “UCSB”

String

0 1 2 3

schoolName

index

- Print the first character

schoolName[0]

- Print the last character

schoolName[3]

index of the last element

schoolName[-1]

schoolName[len(schoolName)-1]

- Print the second character

schoolName[1]

Indexing in strings

Positions of characters in a string start at index 0

schoolName = “UCSB”

```
>>> schoolName[0]
```

"U"

```
>>> schoolName[1]
```

"C"

```
>>> schoolName[2]
```

"S"

```
>>> schoolName[4]
```

Error : Index 4 is out of the valid bounds

Word Play

Write code that produces the following (different) output for the inputs "Diba" and "Eric"

What is your name? Diba
Hi Dibaaaaaa !!!!

What is your name? Eric
Hi Ericccccc !!!!

Python code ...

```
name = input ("What is your name?")
x = "Hi _" + name +
    name[-1] * 5 + "!!!!"

print (x)
```

Extracting substrings

- Also known as slicing!

```
>>> schoolName = “UCSB”
>>> print(schoolName[1:3])
>>> print(schoolName[:3])
>>> print(schoolName[:-1])
```

- Comparison (in, not in):

```
>>>“CS” in schoolName

>>>“CS” not in schoolName
```

More word play

Write code that produces the following (different) output for the inputs “Diba” and “Eric”

Run 1:

What is your name? Diba

Hi Dibaaaaaa !!!!

I meant hi Diiiiiba

Sorry I have a cold, Biba

Run 2:

What is your name? Eric

Hi Ericccccc !!!!

I meant hi Errrrric

Sorry I have a cold, Iric

*Function*ing in Python

```
# my own function!
def dbl( x ):
    """ returns double its input, x """
    return 2x
```

This doesn't look quite right…

*Functio*ning in Python

```
# my own function!
def dbl( x ):
    """ returns double its input, x """
    return 2*x
```

Some of Python's *baggage*…

Docstrings

They become part of python's built-in help system!
With each function be sure to include one that

(1) describes overall what the function does, and
(2) explains what the inputs mean/are

keywords

`def` starts the function
`return` stops it immediately
and sends back the return value

Comments

They begin with #

Essential Definitions and Rules
(do memorize)

parameter (also called argument)

```
# my own function!
def dbl( x ):
    """ returns double its input, x """
    print "Doubling input ", x
    return 2*x
```

comment

function header

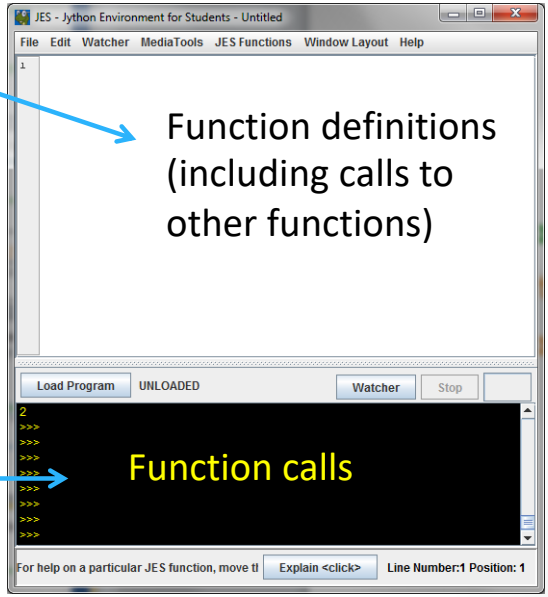
docstring

Function body

Indentation: All the lines in the function body are indented from the function header, and all to the same degree

Flow of Execution

```
# my own function!
def dbl( x ):
    """ returns double its input, x """
    print("Doubling input ", x)
    return 2*x
```



Function definitions (including calls to other functions)

```
>>> dbl( 21 )
```

Function calls

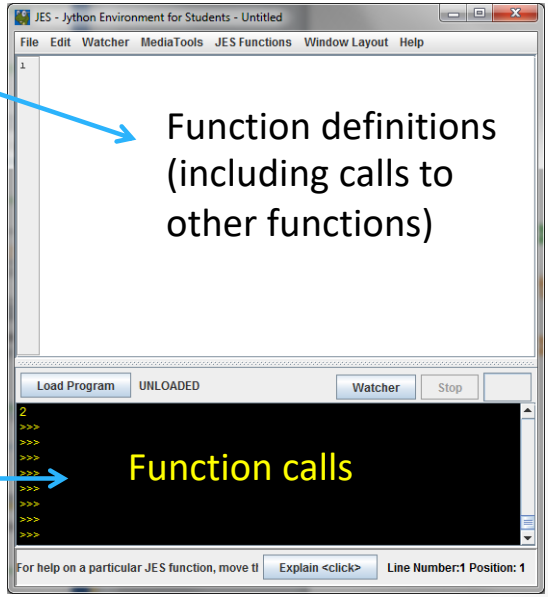
When you call a function, Python executes the function starting at the first line in its body, and carries out each line in order (though some instructions cause the order to change… more soon)

Parameters are special variables

```
# my own function!
def dbl( x ):
    """ returns double its input, x """
    print("Doubling input ", x)
    return 2*x
```

x

```
>>> dbl( 21 )
```



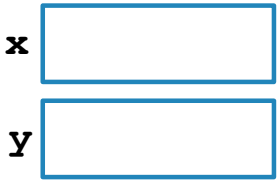
Function definitions (including calls to other functions)

Function calls

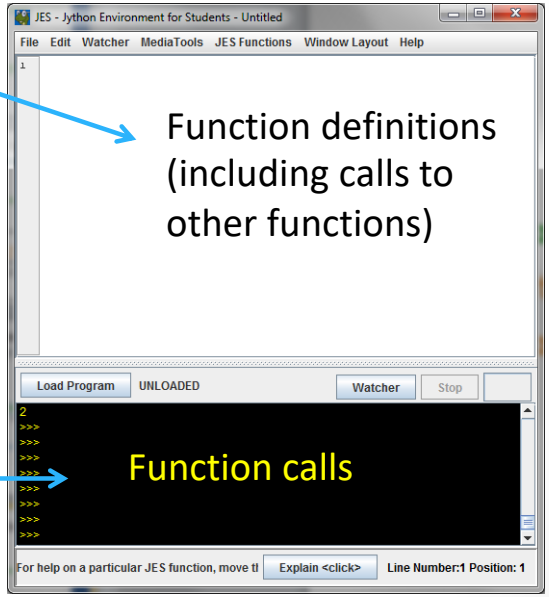
When you call a function, the value you put in parenthesis gets put into the “box” labeled with the name of the parameter and is available for use within the function.

Multiple parameters are allowed

```
# my own function!
def times( x, y ):
    """ returns x times y """
    print("Multiplying ", x, "and", y)
    return x*y
```




```
>>> times( 21, 2 )
```



Function definitions (including calls to other functions)

Function calls

When you call a function, the values you put in parenthesis gets put into the “boxes” labeled with the names of the parameters (in the order in which they are listed)

Which of the following contains a function call?

```
(1)type(4.5)
(2)def dbl(x):
       return 2*x
(3)area(2, 9)
(4)print("Hello")
```

A. (3) only
B. (2) and (3)
C. (1), (3), and (4)
D. All of (1), (2), (3), and (4) include a function call

No parameters is also allowed

```
# my own function!
def fortyTwo( ):
    """ returns 42 """
    return 42
```

```
>>> fortyTwo
```

As much as I like 42, I don't quite like this…

(But you still need parentheses)

```
# my own function!
def fortyTwo( ):
    """ returns 42 """
    return 42
```

```
>>> fortyTwo()
```

Ahh(), much better

No return statement is also allowed

```
# my own function!
def printName( ):
    """ prints a message, no return statement"""
    print("My name is Turtle")
```

```
>>> printName()
```

Functions can call Functions!!

When in doubt, draw it out!

```
def halve( x ):
    """ returns half its input, x """
    return div(x, 2)
def div( y, x ):
    """ returns y / x """
    return y / x
```

```
>>> halve( 84 )
```

Print vs. return

Definition "A"	Definition "B"
`def squared(x):` `  return (x * x)`	`def squared(x):` `  print (x * x)`

Your job: In the following function calls decide which version of squared was used—or whether it is impossible to tell from the output given.

Code	Circle one answer

`>>> squared(7)`
`49`
`>>>`
	A B can't tell

Print vs. return

Definition "A"	Definition "B"
`def squared(x):` `  return (x * x)`	`def squared(x):` `  print (x * x)`

Your job: In the following function calls decide which version of squared was used—or whether it is impossible to tell from the output given.

```
>>> 2 * squared(3)
18
>>>
```

A B can't tell

Definition B returns None which would result in an error
2 * None → error

Functions can call Functions!!

85

```
def halve( x ):
    """ returns half its input, x """
    return div(x, 2)
```

85 → evaluates to 42.5

85 2

```
def div( y, x ):
    """ returns y / x """
    return y / x
```

85/2 = 42.5

```
>>> halve( 85 )
```

42.5

What does halve(85) return?

A. 42
B. 42.5
C. 0
D. 0.02352 (i.e., 2 divided by 85)