

Python Recap

CSC148, INTRODUCTION TO COMPUTER SCIENCE

DIANE HORTON, SADIA SHARMIN & JONATHAN CALVER



Key terms and phrases

object

immutable

id/type/value

mutable

variable

mutate

refers to

re-assign

assignment statement

alias

for loop gotcha

```
lst = [6, 2, 0, 44]
```

```
for item in lst:  
    item = item + 1
```

```
print(lst)
```

Do not assign to the loop variable in a for loop.

The Python Memory Model: Functions

WHAT HAPPENS WHEN WE CALL A FUNCTION?



Example 1

```
def mess_about(n: int, s: str) -> None:  
    message = s * n  
    s = message  
  
if __name__ == '__main__':  
    count = 13  
    word = 'nonsense'  
    mess_about(count, word)
```

Example 1

```
def mess_about(n: int, s: str) -> None:  
    message = s * n  
    s = message
```

```
if __name__ == '__main__':  
    count = 13  
    word = 'nonsense'  
    mess_about(count, word)
```

Example 1

```
def mess_about(n: int, s: str) -> None:
    message = s * n
    s = message

if __name__ == '__main__':
    count = 13
    word = 'nonsense'
    mess_about(count, word)
```

Example 1


```
def mess_about(n: int, s: str) -> None:
    message = s * n
    s = message

if __name__ == '__main__':
    count = 13
    word = 'nonsense'
    mess_about(count, word)
```


Example 1

```
def mess_about(n: int, s: str) -> None:  
    message = s * n  
    s = message
```

```
if __name__ == '__main__':  
    count = 13  
    word = 'nonsense'  
    mess_about(count, word)
```



New terminology

frame

data about a function call (including local variables)

call stack

a collection of the frames of the functions that are
currently running

A complete memory model shows...

On the left, the **call stack**, whose frames show the program's variables and which ids they store

On the right, the **object space**, where all objects are shown (id, type, value)

Function calls and the memory model

When a function is called, its parameters become *aliases* of the arguments passed to the function call.

If you understand this, you can do the rest of the worksheet!



Function gotcha

```
def f(x, ...):  
    x = 10  
    ... # Other code  
  
if __name__ == '__main__':  
    course = 148  
    f(course, ...)
```

Do not assign to a function parameter in hopes of changing the argument.



Reading and writing docstrings

DOCSTRINGS COMMUNICATE PROGRAMMER INTENT—HOW DO WE MAKE SURE THIS COMMUNICATION IS CLEAR?



Function Design Recipe quick recap

1. Examples
2. Header
3. Description
4. Body (implementation)
5. Test

Function Design Recipe quick recap

1. Examples
2. Header
3. Description
4. Body (implementation)
5. Test

Anatomy of a docstring

```
def f(x: int, y: int) -> int:
```

```
    """One-sentence summary.
```

```
    Longer description.
```

```
    >>> # Executable examples
```

```
    """
```



Docstrings as planning

A docstring describes **what** a function does, but not how.

All three parts (summary, longer description, examples) work together in communicating the “what”.

Writing a docstring *before* code sets a goal for you.

Docstrings as contracts

By communicating intent, docstrings make a **promise**:

“This is exactly what my function will do.”

Docstrings must explicitly state *everything* they do: returning, printing, and mutating are all separate effects, and must be clearly stated.

Docstrings as contracts (flip side)

By communicating intent, docstrings make a **promise**:

“This is exactly what my function will do, **as long as you call the function in the way I’ve described.**”

The programmer may make assumptions about the arguments to their function, as long as these assumptions are clearly communicated.



Preconditions

A **precondition** is a property a function's arguments must satisfy to ensure that the function works as described.

Examples:

- Parameter types
- “1st can't be empty”
- “ $n \geq 0$ ”
- “the given file must exist”

Anatomy of a docstring

```
def f(x: int, y: int) -> int:  
    """One-sentence summary.
```

```
    Longer description,  
    including preconditions (other than types).
```

```
>>> # Executable examples  
"""
```

