CSci 127: Introduction to Computer Science

CS @ Hunter College

Frequently Asked Questions

From lecture slips & recitation sections.

- How can I prepare for the paper quizzes (and the final)?
  Starting this week, we'll end with quiz & final practice questions. No new material– focus on problem solving.
  (If you need to leave early, do so before we start so to not disturb your classmates.)
- Do I have to take the final?
  Yes, you have to pass the final (60 out of 100 points) to the pass the class.
- Can I take the course No Credit/Credit?
  Yes, but check with your advisor that it is possible with your major and standing.
- To earn a Credit grade, what do I need?
  - *Final can replace missing lecture slips or quizzes. Programs are 30%.*
  - *You need to pass the final, which takes 60 out of 100 points.*
  - *If final counts 70%, that would be 60% of 70 = 42 points. Need 70 - 42 = 28 points (of 30) on the programs (or 52 programs).*
  - *With higher final score, you need fewer programs: Final: 80, Programs: 27.*
  - *More lecture slips & quizzes help: 10 lectures slips (5%) and 5 quizzes (10%) leave 50% for the final. Passing final with 60% would need 46 programs for credit. 80% on final, need 28 programs...*
  - *Always good to aim a bit higher!*

Today's Topics

- Functions
- Github
- Anna Whitney (Google) & Design Challenge
- Final Exam Overview

Functions

```
#Name:  your name here
#Date:  October 2017
#This program, uses functions,
#    says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any input parameters, surrounded by parenthesis:
  Example: `print("Hello", "World")`
- Can write, or **define** your own functions, which are stored, until invoked or called.

In Pairs or Triples:

Predict what the code will do:

```
#Greet loop example

def greetLoop(person):
    print("Greetings")
    for i in range(5):
        print("Hello", person)

greetLoop("Thomas")
```

```
# From "Teaching with Python" by John Zelle

def happy():
    print("Happy Birthday to you!")

def sing(P):
    happy()
    happy()
    print("Happy Birthday dear " + P + "!")
    happy()


sing("Fred")
sing("Thomas")
sing("Hunter")
```

Python Tutor

```
#Greet loop example

def greetLoop(person):
    print("Greetings")
    for i in range(5):
        print("Hello", person)


greetLoop("Thomas")
```

```
# From "Teaching with Python" by John Zelle

def happy():
    print("Happy Birthday to you!")

def sing(P):
    happy()
    happy()
    print("Happy Birthday dear " + P + "!")
    happy()


sing("Fred")
sing("Thomas")
sing("Hunter")
```

(Demo with pythonTutor)

Input Parameters & Return Values

```python
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip:' ))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip:' ))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```

Formal Parameters

Actual Parameters

- Functions can have **input parameters**.
- Surrounded by parenthesis, both in the function definition, and in the function call (invocation).
- The "placeholders" in the function definition: **formal parameters**.
- The ones in the function call: **actual parameters**.
- Functions can also **return values** to where it was called.

In Pairs or Triples:

Predict what the code will do:

```
def prob4():
    verse = "jam tomorrow and jam yesterday,"
    print("The rule is,")
    c = mystery(verse)
    w = enigma(verse,c)
    print(c,w)
def mystery(v):
    print(v)
    c = v.count("jam")
    return(c)
def enigma(v,c):
    print("but never", v[-1])
    for i in range(c):
        print("jam")
    return("day.")
prob4()
```

```
#Fall 2013 Final Exam, 5

def kuwae( inLst ):
    tot = 1
    for item in inLst:
        tot = tot * item
    return tot

def foo( inLst ):
    if ( inLst[-1] > inLst[0] ):
        return kuwae( inLst )
    else:
        return -1

foo( [2, 4, 6, 8] )

foo( [4002, 328, 457, 1] )
```

Python Tutor

```python
def prob4():
    verse = "jam tomorrow and jam yesterday,"
    print("The rule is,")
    c = mystery(verse)
    w = enigma(verse,c)
    print(c,w)
def mystery(v):
    print(v)
    c = v.count("jam")
    return(c)
def enigma(v,c):
    print("but never", v[-1])
    for i in range(c):
        print("jam")
    return("day.")
prob4()
```

```python
#Fall 2013 Final Exam, 5

def kuwae( inLst ):
    tot = 1
    for item in inLst:
        tot = tot * item
    return tot

def foo( inLst ):
    if ( inLst[-1] > inLst[0] ):
        return kuwae( inLst )
    else:
        return -1

foo( [2, 4, 6, 8] )

foo( [4002, 328, 457, 1] )
```

(Demo with `pythonTutor`)

Input Parameters

```python
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip:' ))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip:' ))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```

Formal Parameters

Actual Parameters

- When called, the actual parameter values are copied to the formal parameters.
- All the commands inside the function are performed on the copies.
- The actual parameters do not change.
- The copies are discarded when the function is done.
- The time a variable exists is called its **scope**.

Input Parameters: What about Lists?

```
#Fall 2013 Final Exam, 5

def kuwae( inLst ):
    tot = 1
    for item in inLst:
        tot = tot * item
    return tot

def foo( inLst ):
    if ( inLst[-1] > inLst[0] ):
        return kuwae( inLst )
    else:
        return -1

foo( [2, 4, 6, 8] )

foo( [4002, 328, 457, 1] )
```

- When called, the actual parameter values are copied to the formal parameters.

- What is copied with a list?

- The address of the list, but not the individual elements.

- The actual parameters do not change, but the inside elements might.

- Easier to see with a demo.

Python Tutor

```
#Fall 2013 Final Exam, 5

def kuwae( inLst ):
    tot = 1
    for item in inLst:
        tot = tot * item
    return tot

def foo( inLst ):
    if ( inLst[-1] > inLst[0] ):
        return kuwae( inLst )
    else:
        return -1

foo( [2, 4, 6, 8] )

foo( [4002, 328, 457, 1] )
```

(Demo with `pythonTutor`)

In Pairs or Triples:

```python
def bar(n):
    if n <= 8:
        return 1
    else:
        return 0

def foo(l):
    n = bar(l[-1])
    return l[n]
```

- What are the formal parameters for the functions?

- What is the output of:
  ```python
  r = foo([1,2,3,4])
  print("Return: ", r)
  ```

- What is the output of:
  ```python
  r = foo([1024,512,256,128])
  print("Return: ", r)
  ```

Python Tutor

```
def bar(n):
    if n <= 8:
        return 1
    else:
        return 0

def foo(l):
    n = bar(l[-1])
    return l[n]
```

(Demo with `pythonTutor`)

In Pairs or Triples:

```
def prob4(amy, beth):                def helper(meg,jo):
    if amy > 4:                          s = ""
        print("Easy case")              for j in range(meg):
        kate = -1                           print(j, ": ", jo[j])
    else:                                   if j % 2 == 0:
        print("Complex case")                   s = s + jo[j]
        kate = helper(amy,beth)                 print("Building s:", s)
    return(kate)                        return(s)
```

- What are the formal parameters for the functions?

- What is the output of:

      r = prob4(4,"city")
      print("Return: ", r)

- What is the output of:

      r = prob4(2,"university")
      print("Return: ", r)

Python Tutor

```python
def prob4(any, beth):
    if any > 4:
        print("Easy case")
        kate = -1
    else:
        print("Complex case")
        kate = helper(any,beth)
    return(kate)
```

```python
def helper(meg,jo):
    s = ""
    for j in range(meg):
        print(j, ": ", jo[j])
        if j % 2 == 0:
            s = s + jo[j]
            print("Building s:", s)
    return(s)
```

(Demo with pythonTutor)

Github

Octocat

- Like Google docs for code...
- Used to share code, documents, etc.
- More formally: `git` is a version control protocol for tracking changes and versions of documents.
- Github provides hosting for repositories (**'repos'**) of code.
- Also convenient place to host websites (i.e. `stjohn.github.io`).
- In lab, we will set up github accounts and copy (**'clone'**) documents from the class repo. (More in future courses.)

Recap

```
#Name:  your name here
#Date:  October 2017
#This program, uses functions,
#     says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- On lecture slip, write down a topic you wish we had spent more time (and why).

- Functions are a way to break code into pieces, that can be easily reused.

- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:
  Example: `print("Hello", "World")`

- Can write, or **define** your own functions, which are stored, until invoked or called.

- Functions can have **input parameters** that bring information into the function,

- And **return values** that send information back.

- Both input parameters and return values are optional.