Chapter 3 ICE

1. for n in range(6):

2. The easiest way is to click on the kill window X in the tab of the editor window.

3. In Python 3- 180.0 and 6.75 for the two expressions, in Python 2- 6/18=0! division by zero.

4. Change the 6 to a 6.0 in Python 2.

5. 7 in both Python 3 and python2.

6. Remainder or modulo

7. True, all relations are correct

False, 6 is not greater than 71

0, What’s in ( ) is evaluated first, thus zero(False) divided by anything equal 0

0, What’s in ( ) is evaluated first, thus zero(False) divided by anything equal 0

8. x is something else

9. Nothing prints out

10. x is something else

11. The error reads- RuntimeError: maximum recursion depth exceeded in cmp. cmp is a Python function that discriminates to only allow integers forward in the calculations. You can’t use a Real number to find its factorial!

12. Use the gamma function on the Real number + 1, in our case 3.72. Such as-

>>>**import math**

>>>**math.gamma(3.72)**

4.269416975478383

>>>

13. >>>**print int(x1[0]) + int(x1[1]) + int(x1[2])**

14. newline

15. You can open the file, read from it, and print x1, but x1 will have a single object, 23 33 43 54\n in it, basically a character string of 4 numbers and a newline, and none of the rest of the commnds in the Example will work on that object!

16. Yes, by specifying the objects as int(x1[2]) and int(x1[3]).

17. x1[2]

18. 'programmingprogrammer\nprograms'

19. newline is a character

20. 0

21. Empty set

22. Extracts the whole string.

23. ‘ro’ because 1:3 extracts from the second character to the fourth character, not including the fourth.

24. From the assignments and operations above that line, there are no elements common to r and a.

25. x.add takes only one argument. x.update will only add one 5.0.

26. No, it is an immutable object. For example-

>>> s = (34,35,36,37)

>>> s[1:2] = [-1,-2]

Traceback (most recent call last):

File "<pyshell#5>", line 1, in <module>

s[1:2] = [-1,-2]

TypeError: 'tuple' object does not support item assignment

>>>

27. >>>import regen

28. >>>x = regen.abc()

29. >>>next(x)

1

>>>next(x)

2

>>>next(x)

3

>>>next(x)

‘ijk’

>>>next(x)

‘deff’

>>>next(x)

‘abc’

>>>next(x)

Traceback(most recent call last)

File “<stdin>”, line 1, in <module>

StopIteration

>>>

30. a) in a file named grep.py-

def grepper(template):

print(‘Searching for ‘ + template)

try:

while True:

x = (yield)

id template in x:

print x

except GeneratorExit:

print “Done”

b)

>>>import grep

>>>s = grep.grepper(‘1201’)

c)

>>>s.next()

Searching for 1201

>>>s.send(‘0120 1201 1020’)

0120 1201 1020

>>>s.send(‘3012 3013 3212’)

>>>s.send('12010203101213012')

12010203101213012

>>>

31. a)

def read(text,next\_coroutine):

for line in text.split():

next\_coroutine.send(line)

next\_coroutine.close()

b) assumes grepper module or function is still loaded!

>>>import reader

>>>text2 = “ Python is the most Pythonic enterprise a Pythonista can practice”

>>>found2 = grep.grepper(“Python”)

c)

>>>found2.next()

Searching for Python

>>>reader.read(text2,found2)

Python

Pythonic

Pythonista

Done

>>>

32. SystemExit – sys.exit() function

ArithmeticError – numeric calculations error

EOFError – end of file reached

ImportError – import statement error

KeyError – dictionary key not found

33. Add a new line 3, L = [1,2,3,4,5,6,7,8,9,10]

On new line 4, handler.write(str(L))

34. Traceback NameError name ‘b’ is not defined, b is a local variable in the module arith1.

35. Traceback NameError name ‘arith1’ is not defined, del has erased the module from Python memory.

36. You get no values, because the functions are not bringing to or associating values with the named objects at the Python command line level( which can be called the top level, or the highest module).

37. /bin/sh name , where name is the file that contains the Bourne code.

38. os.path.exists(path)

39. Assuming no one has a /usr/bin/yyy directory, the else statement is executed.

40. words[0], words[1], words[2:]

41. r.split(‘ ‘) , ‘ ‘.join(words

42. No answer required.

43. No answer required.

44. In Tkinter, you can position widgets precisely on the screen by using the *place* geometry manager or by setting the widget's position using the *geometry* method. An example of each is as follows-

**import tkinter as tk**

**root = tk.Tk()**

**root.title("Placing Widgets")**

**# Create a label and place it at coordinates (100, 50)**

**label = tk.Label(root, text="Widget placed at (100, 50)")**

**label.place(x=100, y=50)**

**root.mainloop()**

**import tkinter as tk**

**root = tk.Tk()**

**root.title("Setting Widget Position")**

**# Set the geometry to position the widget at (100, 50) with a width of 200 and height of 100**

**root.geometry("200x100+100+50")**

**label = tk.Label(root, text="Widget with custom position")**

**label.pack()**

**root.mainloop()**

45. Indeterminate repetition, or the While loop.

46. No answer required.

47. Methods are basically an OOP operation on objects, classes, and instances, and functions are modularized, callable-code that declare a procedure to perform in a non-OOP manner, such as in a procedural/functional programming language like C.

48. Numbering starts in the upper left hand corner at 0,0, and proceeds down rows and to the left in columns. So the grid position 0,2 would translate to the zeroth( or top-most) row, the third column to the right.

49. You can import sys, and place a Button in grid cell 0,2, for example, that calls sys.exit().

50. In the function child, after the print statement.

51. This illustrates the most important aspect of running threads concurrently, whether in Python or at the system programming level. There is no guarantee that, because of the scheduling of threads in the parent process, that the order of the **myId** variable will be printed exactly the same on each loop. That is the reason you run them concurrently.

52. Same answer as 48.

53. In Block 3., in the for loop before the **time.sleep(GoToSleep)** line.

54. Because the scheduling of the output to stdout from the 2 threads is not entirely determinate, sometimes Producer runs for awhile, and then Consumer runs, etc.. This is an important characteristic of multi-threaded programs.