COMP 150:
Basics of Computer Organization
Week 04, Lab 03

Agenda

Big pile of practice: variables, functions, conditionals, and loops

1. Review
2. Practice Notebook

Review: Variables, lists, dictionaries

```
x = 15

y = “banana”

z = True

l = [“lists”, “can”, “contain”, “lots of things”, 5, 20, -1]

d = {“dictionaries”: 15,
      “map” : “twenty”,
      “keys”: [“a”,”b”],
      “to”  : 19,
      “values”     : 105,
}
```

Review: Lists

```
l = [5, 20, -1]

# No need for them to contain the same datatypes, but we can
do
# more if they do

dir(l) # Useful - tells us everything we can do!

len(l)
sum(l)
sort(l)
reverse(l)
```

Review: Dictionaries (Dicts)

```
d = {“plates”: 15,
      “cups”: 20,
      “serviettes”: 4,
}

# Dicts map keys to values
# Keys can be anything single-valued (“a”, 1, “apple”, “apple_pie”)
# Values can be anything single-value as well, but it’s common to
store lists or more dicts as values

d[“plates”]
d.keys()
d.values()
```

Review: Conditionals, And/Or

```
# Everything below this evaluates to True

True
10 > 5
True and True
(10 > 5) and True
(10 > 5) or False
(10 > 5) and not (10 < 2)
“x” in “string_containing_x”
5 in [2,3,4,5]
```

Review: If/Elif/Else

```
if True:
	# This will definitely run

if not False:
	# This will also run


if sun_shining == True:
	# Don’t turn on the lights
else:
	# Turn them on


if sun_brightness > 100:
	# Don’t turn on the lights
elif sun_brightness > 50:
	# Turn on the lights just a little
else:
	# Turn the lights on all the way
```

Review: While

```
while True:
	# This will run for forever

while x < 10:
	# Do something that changes x

# You can also manually break the while-loop
while True:
	# Do some stuff
	if something_happens == True:
		break  # This will exit the loop
```

Review: For

```
# For loops in Python let us count over something

for x in [1,2,3,4,5]:
      print(x * 10)

for i in range(0, 10):
      print(i)

for key in dict.keys():
      dict[key] = something_new
```

What's with all the tabs??

Python uses *whitespace denotation* to separate code into logical pieces

```
def fizz_buzz(n):
    result = []

    for i in range(1, n + 1):
        if i % 3 == 0 and i % 5 == 0:
            # Add "FizzBuzz" to the result list
            result.append("FizzBuzz")
        elif i % 3 == 0:
            # Add "Fizz" to the result list
            result.append("Fizz")
        elif i % 5 == 0:
            # Add "Buzz" to the result list
            result.append("Buzz")
        else:
            # Add the current number as a string to the result list
            result.append(str(i))
    return result


def other_function():
       x = “banana”
```

What's with all the tabs??

Python uses *whitespace denotation* to separate code into logical pieces

```
def fizz_buzz(n):
    result = []

    for i in range(1, n + 1):
        if i % 3 == 0 and i % 5 == 0:
            # Add "FizzBuzz" to the result list
            result.append("FizzBuzz")
        elif i % 3 == 0:
            # Add "Fizz" to the result list
            result.append("Fizz")
        elif i % 5 == 0:
            # Add "Buzz" to the result list
            result.append("Buzz")
        else:
            # Add the current number as a string to the result list
            result.append(str(i))
    return result
```

```
def other_function():
      x = “banana”
```

What's with all the tabs??

Python uses *whitespace denotation* to separate code into logical pieces

```
def fizz_buzz(n):
    result = []

    for i in range(1, n + 1):
        if i % 3 == 0 and i % 5 == 0:
            # Add "FizzBuzz" to the result list
            result.append("FizzBuzz")
        elif i % 3 == 0:
            # Add "Fizz" to the result list
            result.append("Fizz")
        elif i % 5 == 0:
            # Add "Buzz" to the result list
            result.append("Buzz")
        else:
            # Add the current number as a string to the result list
            result.append(str(i))
    return result


def other_function():
       x = “banana”
```

What's with all the tabs??

Python uses *whitespace denotation* to separate code into logical pieces

```
def fizz_buzz(n):
    result = []

    for i in range(1, n + 1):
        if i % 3 == 0 and i % 5 == 0:
            # Add "FizzBuzz" to the result list
            result.append("FizzBuzz")
        elif i % 3 == 0:
            # Add "Fizz" to the result list
            result.append("Fizz")
        elif i % 5 == 0:
            # Add "Buzz" to the result list
            result.append("Buzz")
        else:
            # Add the current number as a string to the result list
            result.append(str(i))
    return result


def other_function():
      x = “banana”
```

What's with all the tabs??

Python uses *whitespace denotation* to separate code into logical pieces

```
def fizz_buzz(n):
    result = []

    for i in range(1, n + 1):
        if i % 3 == 0 and i % 5 == 0:
            # Add "FizzBuzz" to the result list
            result.append("FizzBuzz")
        elif i % 3 == 0:
            # Add "Fizz" to the result list
            result.append("Fizz")
        elif i % 5 == 0:
            # Add "Buzz" to the result list
            result.append("Buzz")
        else:
            # Add the current number as a string to the result list
            result.append(str(i))
    return result


def other_function():
      x = “banana”
```

What's with all the tabs??

Python uses *whitespace denotation* to separate code into logical pieces

```
def fizz_buzz(n):
    result = []

    for i in range(1, n + 1):
        if i % 3 == 0 and i % 5 == 0:
            # Add "FizzBuzz" to the result list
            result.append("FizzBuzz")
        elif i % 3 == 0:
            # Add "Fizz" to the result list
            result.append("Fizz")
        elif i % 5 == 0:
            # Add "Buzz" to the result list
            result.append("Buzz")
        else:
            # Add the current number as a string to the result list
            result.append(str(i))
    return result


def other_function():
       x = “banana”
```

What's with all the tabs??

Python uses *whitespace denotation* to separate code into logical pieces

```
def fizz_buzz(n):
    result = []

    for i in range(1, n + 1):
        if i % 3 == 0 and i % 5 == 0:
            # Add "FizzBuzz" to the result list
            result.append("FizzBuzz")
        elif i % 3 == 0:
            # Add "Fizz" to the result list
            result.append("Fizz")
        elif i % 5 == 0:
            # Add "Buzz" to the result list
            result.append("Buzz")
        else:
            # Add the current number as a string to the result list
            result.append(str(i))
    return result


def other_function():
      x = “banana”
```

What's with all the tabs??

Python uses *whitespace denotation* to separate code into logical pieces

```
def fizz_buzz(n):
    result = []

    for i in range(1, n + 1):
        if i % 3 == 0 and i % 5 == 0:
            # Add "FizzBuzz" to the result list
            result.append("FizzBuzz")
        elif i % 3 == 0:
            # Add "Fizz" to the result list
            result.append("Fizz")
        elif i % 5 == 0:
            # Add "Buzz" to the result list
            result.append("Buzz")
        else:
            # Add the current number as a string to the result list
            result.append(str(i))
    return result


def other_function():
       x = “banana”
```

What's with all the tabs??

Python uses *whitespace denotation* to separate code into logical pieces

```
def fizz_buzz(n):
    result = []

    for i in range(1, n + 1):
        if i % 3 == 0 and i % 5 == 0:
            # Add "FizzBuzz" to the result list
            result.append("FizzBuzz")
        elif i % 3 == 0:
            # Add "Fizz" to the result list
            result.append("Fizz")
        elif i % 5 == 0:
            # Add "Buzz" to the result list
            result.append("Buzz")
        else:
            # Add the current number as a string to the result list
            result.append(str(i))
    return result
```

```
def other_function():
      x = “banana”
```

Lab 03

1. Download the .ipynb from Sakai
2. Complete the tasks (make sure it all runs!)
3. Save the .ipynb, put your LUC username in the filename
4. Upload your version to the assignment on Sakai
5. Submit