CSCi 1012 [Section 10]

Introduction to Programming with Python

Prof. Kartik Bulusu, CS Dept.

Course start date January 17, 2024
Lecture location 1957 E street Room 213
Lecture times Monday, 3:45 PM to 5:00 PM

Wednesday-lab
3:45 PM to 5:00 PM
Section-30: MON 352
Section-31: SEH 4040
Section-34: TOMP 310
Section-35: TOMP 204

Friday-lab
3:45 PM to 5:00 PM
Section-32: SEH 4040
Section-33: TOMP 309
Section-36: TOMP 306
Section-37: TOMP 107

GW School of Engineering & Applied Science
THE GEORGE WASHINGTON UNIVERSITY

Spring 2024

Photo: Kartik Bulusu

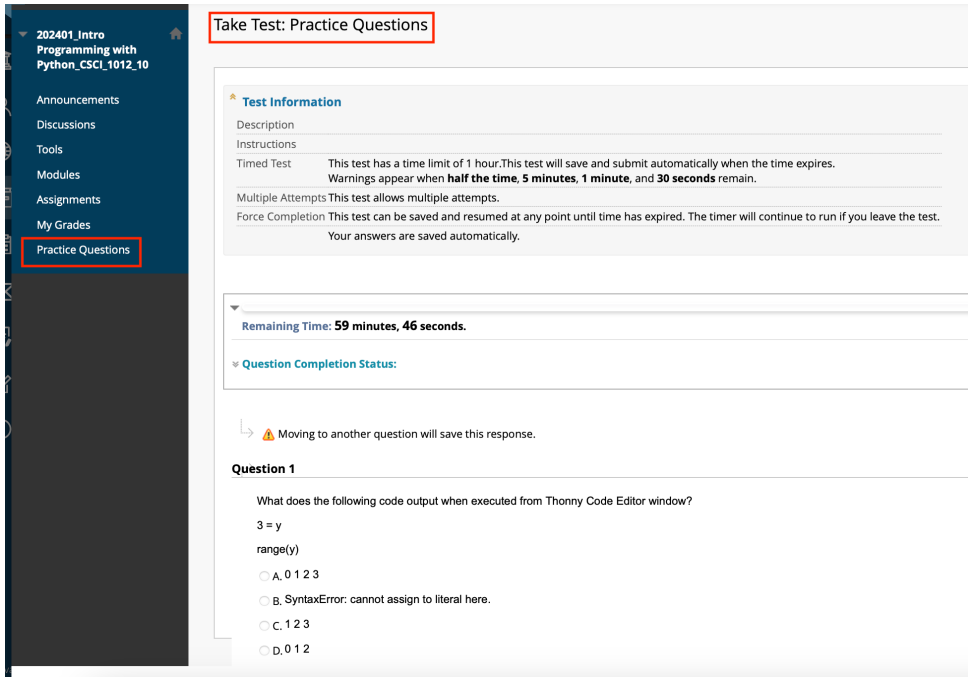
Class Policy on Collaboration

- **You may *not* *discuss Modules, Assignments, Quizzes and Exams* among yourselves.**
- **Each student is expected to work out the course deliverables *independently*.**
- **Under *no circumstances* may you look at another student's *Modules, Assignments, Quizzes and Exams*, or look for answers to *Modules, Assignments, Quizzes and Exams* anywhere other than in the text in the course website.**
- **You are encouraged to discuss the class material on Ed-discussion board or in-person with the instruction team.**
- **You may *not* discuss *Modules, Assignments, Quizzes and Exams* nor give out hints for the same on problems on the Ed-discussion board or with other students in-person.**

All violations will be treated as violations of the Code of Academic Integrity.

Announcements

- Exam on Monday, 4/15 during the lecture.
  - Paper exam
  - Closed everything (no notes, no Thonny)
  - Multiple choice/ fill in the blanks/ write the output – similar to quizzes
  - 25 questions
  - 100 points
- 48-hour extensions:
  - After the 4th extension your subsequent late submissions will not be considered toward final grade.
- 15-20 min info session on minoring in CS
  - On 04/08 during lecture
- Practice questions for you to review is created on Blackboard



HWs
- Due dates
- Late work
- Extensions

Date	Topic(s)	Wednesday Lab Date	Friday Lab Date	Assignment(s)
Week 11 [04/01/2024]	`while` loops, I/O	04/03/2024	04/05/2024	Unit 1 » Module 4 & Module 5 (Due **April 08, 2024** by **11:59 PM**)
Week 13 [04/15/2024]	**Examination**	04/17/2024	04/19/2024	Unit 2 » Module 0 & Module 1 (Due **April 22, 2024** by **11:59 PM**)

- **IMPORTANT:** Please attend the ONLY lab that you registered into.

Late Work

- **Late work is not accepted, with the following exceptions:**
  - Every student many turn in **as many as four (in total, not each) assignments or modules 48 hours after the deadline with no penalty**. Requesting an extension is not necessary.
- **Extensions** will be **granted should there arise circumstances beyond your control** that impede your ability to complete coursework.
  - Notify your professor as soon as feasible in these cases.
    - Examples of such circumstances include (but are not limited to) illness, death in the family, and loss of housing. To ensure fairness toward all students, we will request documentation of such circumstances.

Submission Tips

- Pay attention to file names

extra spaces

assignment1.zip ≠ ***assignment 1.zip***

caesar_shift.py ≠ ***caesar_shift .py***

extra spaces

- Pay attention to small details

```
x is 3 and j is 2, x + j = 5
x is 3 and j is 3, x + j = 6
x is 3 and j is 4, x + j = 7
x is 3 and j is 5, x + j = 8
x is 3 and j is 6, x + j = 9
x is 3 and j is 7, x + j = 10
x is 3 and j is 8, x + j = 11
```

≠

missing commas

```
x is 3 and j is  2 x + j 5
x is 3 and j is  3 x + j 6
x is 3 and j is  4 x + j 7
x is 3 and j is  5 x + j 8
x is 3 and j is  6 x + j 9
x is 3 and j is  7 x + j 10
x is 3 and j is  8 x + j 11
```

extra space *missing =*

- Don't add extra endline spaces

print("++++", end = "") →

```
++++
+  +
+  +
++++
```

→ ***What Auto Grader Sees…***

'++++\n+ +\n+ +\n+ +\n++++'

≠

print("++++", end = " ") →

```
++++ 
+  + 
+  + 
++++ 
```

→ '++++ \n+ +\n+ +\n+ +\n++++ '

extra spaces

Skeleton of the for-loop

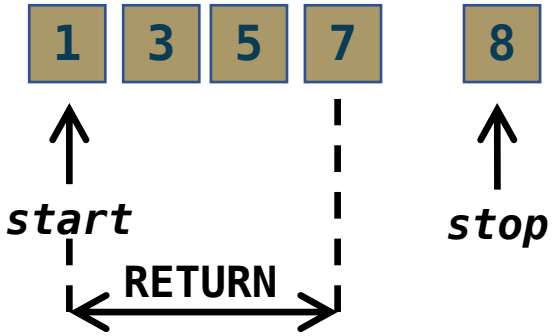
`range(stop)`

`range(start, stop)`

`range(start, stop, step)`

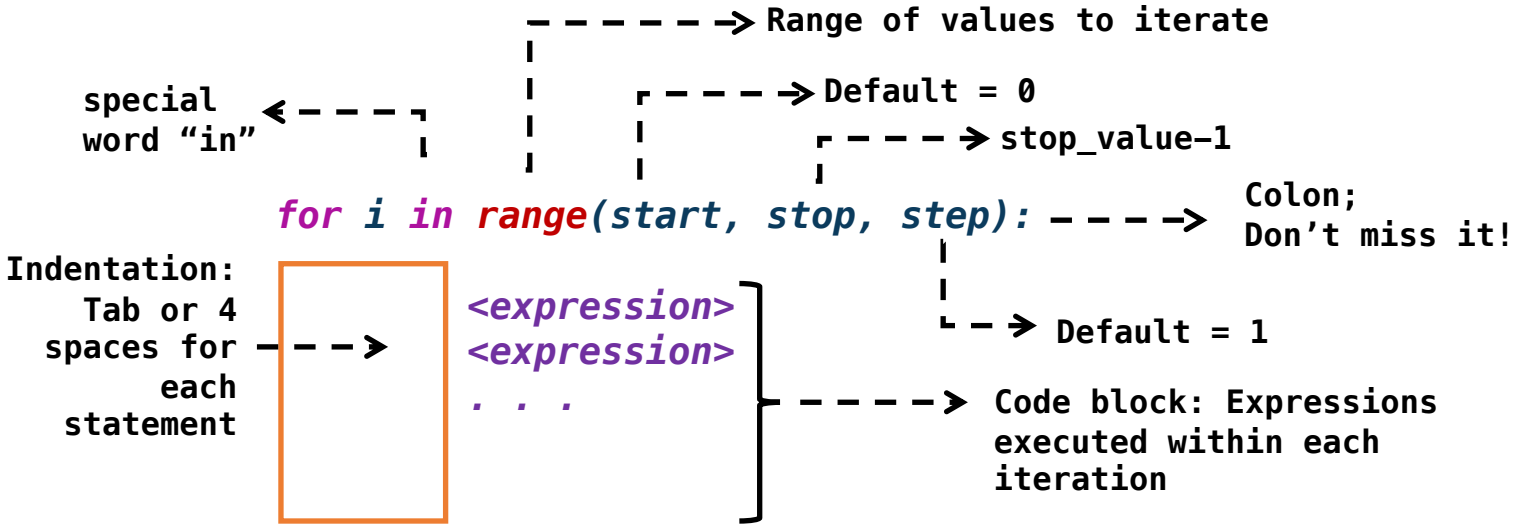
```
start:      at the value (default = 0)
step:       up or down at the increment value (default = 1)
stop:       at the value but not including it
```

```
>>> range(1,8,2)
```




```
for i in range(start, stop, step):

        <expression>
        <expression>
        . . .
```



Note:
Iteration over a `list` is also possible in `for-loops`

How can we add “tests” to our code ?

Sources:
Branching and Iterations By Ana Bell: https://youtu.be/0jljZRnHwOI
Guttag, John, Introduction to Computation and Programming Using Python: With Application to Understanding Data (The MIT Press) second edition

Assume `i` and `j` are variables of `int`, `float` and `string` type.

We can test using logic operators

`i > j`	Greater than
`i >= j`	Greater than or equal to
`i < j`	Less than
`i <= j`	Less than or equal to
`i == j`	Equality
`i != j`	Inequality

`True,` if i and j are same.

`True,` if i and j are not the same.

Note:
= is an assignment
== is a test

Comparisons evaluate to a Boolean

- `True`
- `False`

You are allowed to compare

- `int` with `int`
- `float` with `float`
- `int` and `float`
- `string` with `string`

But not a number with strings.

String comparisons are lexicographical
- **Follows what comes first in the alphabet**

Skeleton of the control flow - branching using `if-construct`

Decision path

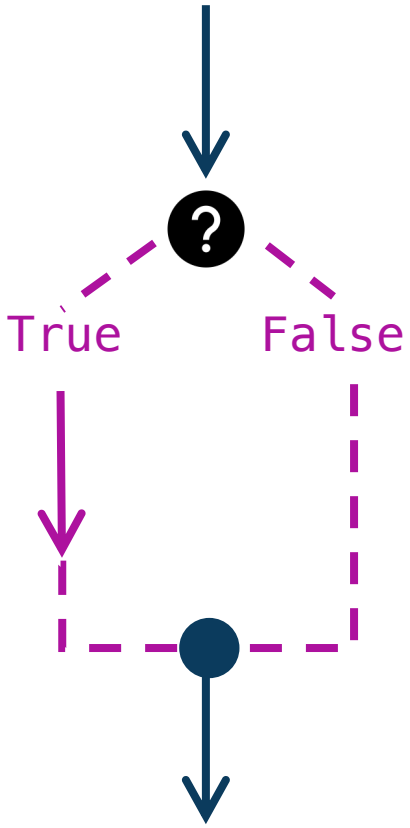
Instructions/ Expressions

```
<instructions>
<instructions>
<instructions>
. . .
```

```
if <condition>:

        <expression>
        <expression>
        . . .
```

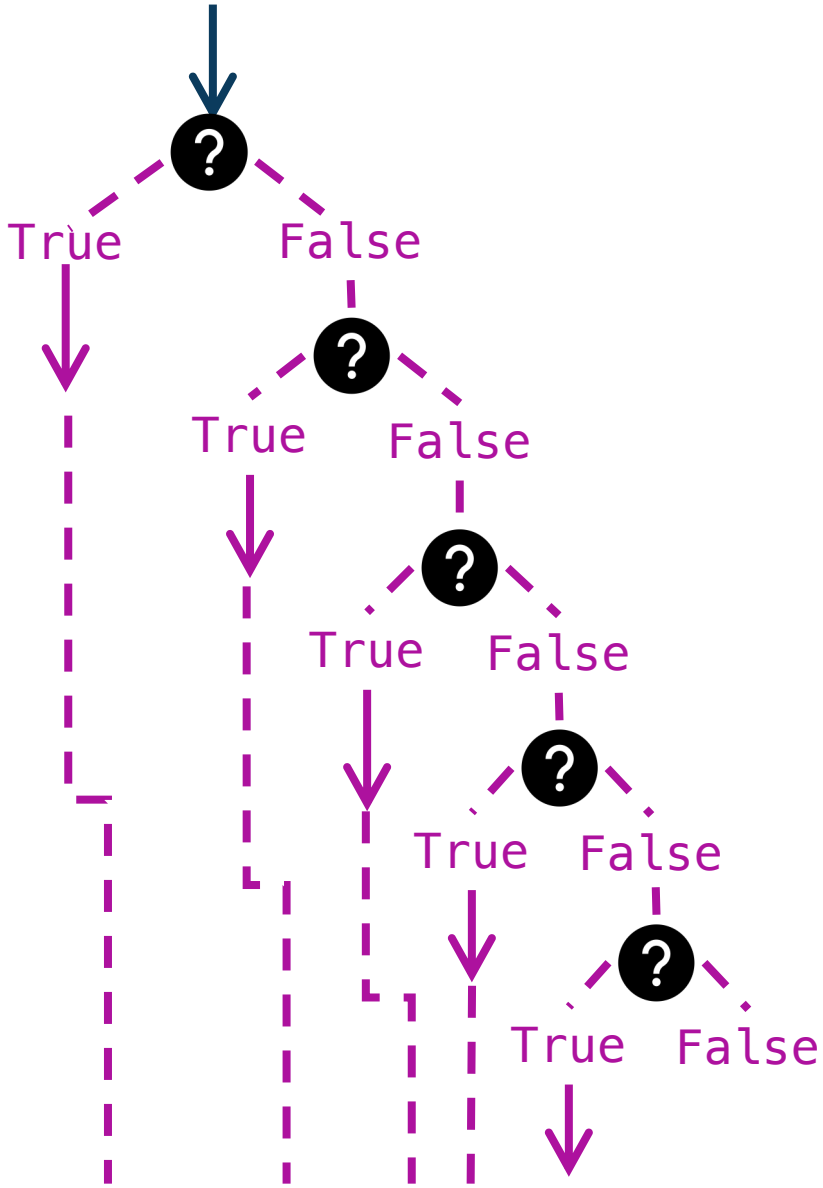
```
<instructions>
<instructions>
<instructions>
. . .
```




```
if <condition>:
    <expression>
    <expression>
    . . .
```

- `if` → special word "if"
- `<condition>` → `<condition>` that gets evaluated to `True` or `False`
- `:` → Colon; Don't miss it!
- Indentation: Tab or 4 spaces for each statement
- `<expression>` … → Code block: Evaluate expressions if the `<condition>` is `True`.

Source:
https://en.wikipedia.org/wiki/Cookie#/media/File:2ChocolateChipCookies.jpg
https://en.wikipedia.org/wiki/Elliott_School_of_International_Affairs#/media/File:Elliott_Entrance.JPG




```
<instructions>
. . .
if <condition>:
        <expression>
        <expression>
        . . .
elif <condition>:
        <expression>
        <expression>
        . . .
elif <condition>:
        <expression>
        <expression>
        . . .
elif <condition>:
        <expression>
        <expression>
        . . .
        .
        .
else:   .
        <expression>
        <expression>
        . . .
```

if-elif-else construct

Is there an alternative when there are too many elif-statements ?

Skeleton of the control flow - branching using `while-loop`

Decision path

Instructions/
Expressions

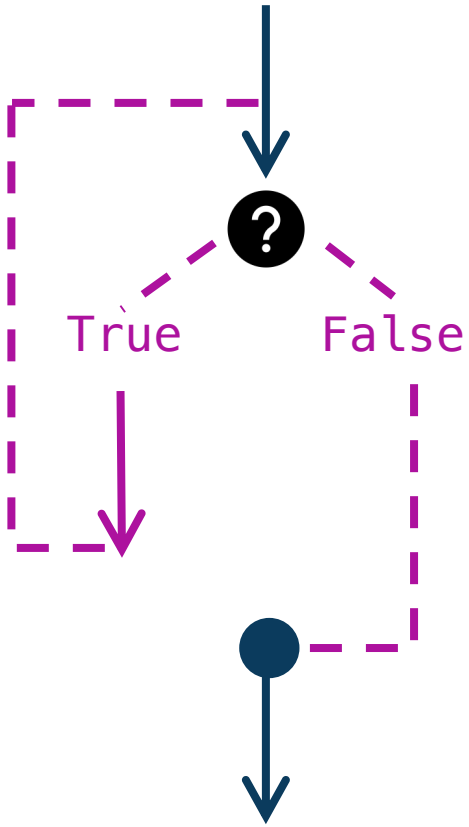
```
<instructions>
<instructions>
<instructions>
. . .

<Initialization>

while <condition>:

        <expression>
        <expression>
        . . .

<instructions>
<instructions>
<instructions>
. . .
```




```
while <condition>:
```

special word "while"

<condition> that gets evaluated to True or False

Colon; Don't miss it!

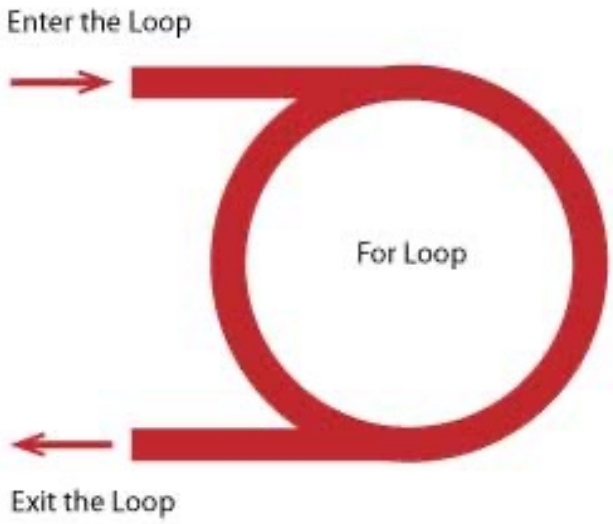
Indentation: Tab or 4 spaces for each statement

```
        <expression>
        <expression>
        . . .
```

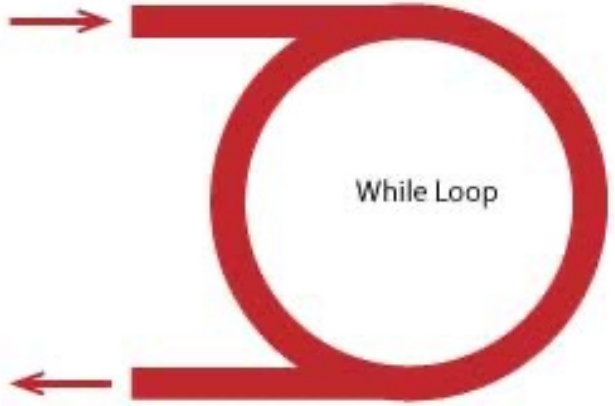
Code block: Evaluate expressions if the <condition> is True.

Note:
One exception to the <Initialization> for a `while-loop` is "`while True:`"

(For) How many times do you want to execute a piece of code ?



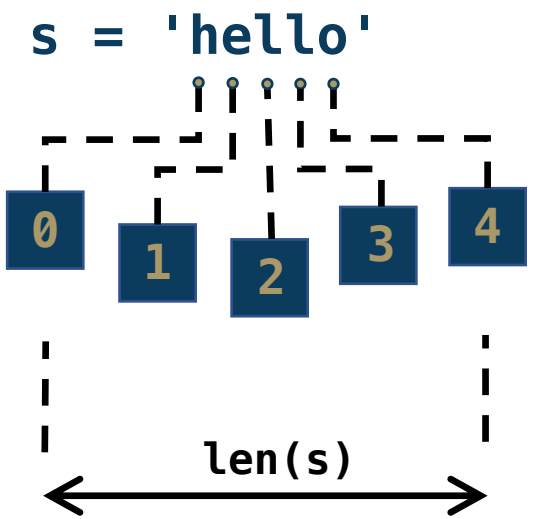
(While) Some condition is met, Enter the Loop



Exit the Loop, when the condition is no longer met

Source:
https://www2.seas.gwu.edu/~cs4all/1012/unit1/module1.4.html

for-loop vs while-loop




```
s = 'hello'
k = len(s) - 1
while k >= 0:
    print(s[k])
    k = k - 1
```

while-loop

- `s = 'hello'` → *<instructions>*
- `k = len(s) - 1` → *<Initialization>*
- `while k >= 0:` → *while <condition>*
- `print(s[k])` → *<expression>*
- `k = k - 1` → *<expression>*

```
for k in range(len(s)-1, -1, -1):
    print(s[k])
```

- `range(len(s)-1, -1, -1)` → Range of values to iterate
- `0` → Default = 0
- `-1` → stop_value-1
- `1` → Default = 1
- `print(s[k])` → *<expression>*

break out of a for-loop or while-loop

Source: https://www2.seas.gwu.edu/~cs4all/1012/unit1/module1.4.html

```
for k in range(1, 50):
    if (k+1)*(k+1) > 50:
        print(k)
        break
```

for-loop code block

if-conditional expressions: Will evaluate when "True"

"break" terminates the for-loop

```
k = 1
while True:
    if k*k > 50:
        break
    k = k + 1

k = k - 1
print(k)
```

if-conditional expressions: Will evaluate when "True"

while-loop code block

"break" terminates the while-loop

for-loop within a while-loop

Source: https://www2.seas.gwu.edu/~cs4all/1012/unit1/module1.4.html

"break" terminates the for-loop

```
m = 10
while m <= 10000:
    for k in range(1, m):
        if (k+1)*(k+1) >= m:
            print('largest square <', m, ':',k*k)
            break
    m = m * 10
```

for-loop code block

if-conditional expressions: Will evaluate when "True"

while-loop code block

File I/O in Python

Try the following on the shell window:

```
>>> help(open)
>>> help(str)
>>> dir(str)
```

Source:
https://www2.seas.gwu.edu/~cs4all/1012/unit1/module1.4.html

```
lines = []
with open('testfile.txt', 'r') as in_file:
    line = in_file.readline()
    while line != '':
        lines.append(line.strip())
        line = in_file.readline()

print(type(lines))
print(lines)
```

```
with open('testcopy.txt', 'w') as out_file:
    for line in lines:
        out_file.write(line + '\n')
```

```
data = []
with open('data.txt', 'r') as in_file:
    line = in_file.readline()
    while line != '':
        s = line.strip()
        x = float(s)
        data.append(x)
        line = in_file.readline()

print(data)
```

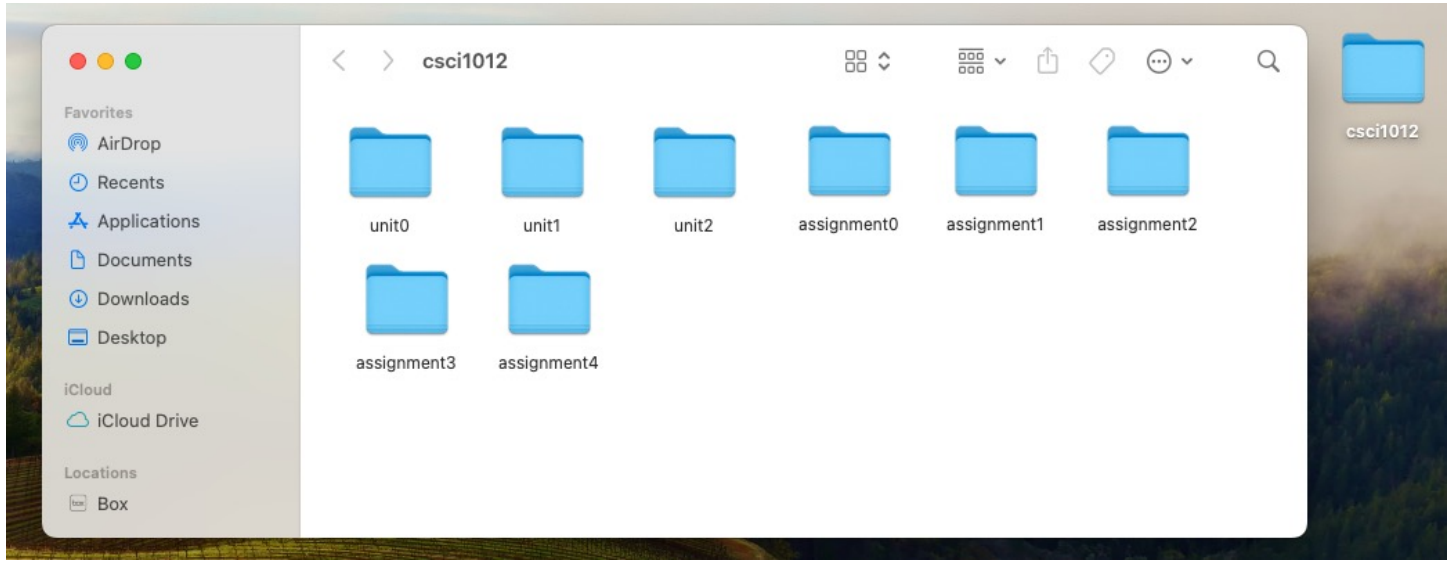
File-folder-structure

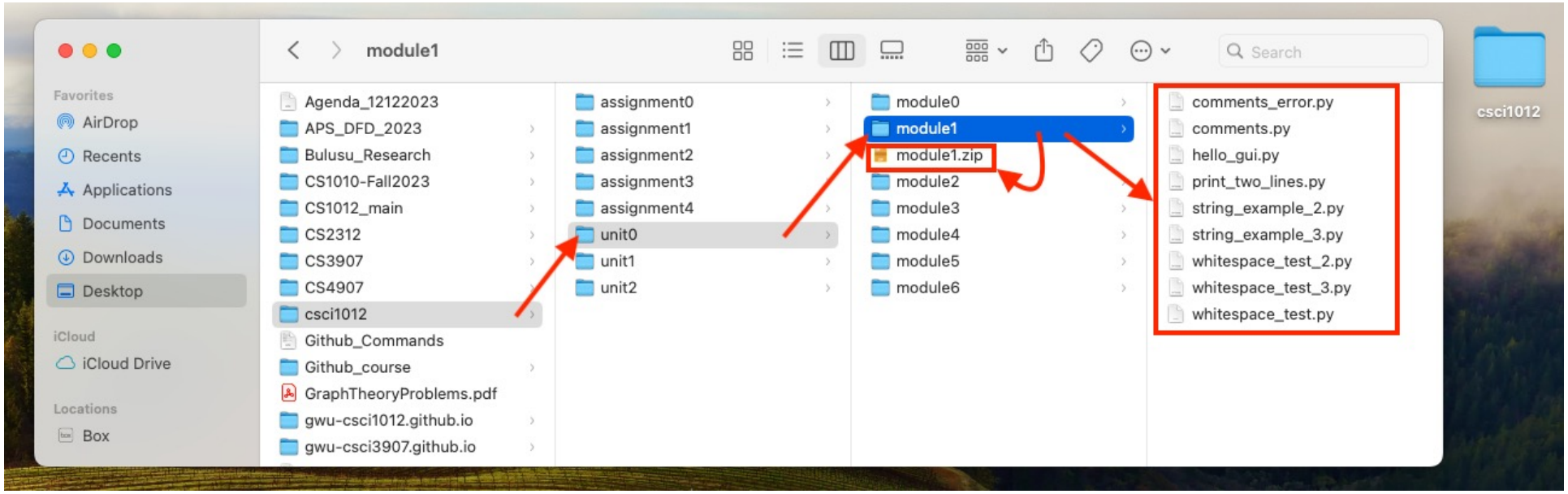
`module0.zip` (correct)

`Module0.zip` (wrong: starts with uppercase)

`module 0.zip` (wrong: space before 0)

`module0.docx` (wrong: not a zip).





See you all in the Wednesday and Friday Labs!