Computer Laboratory 01

CSCI 1913: Introduction to Algorithms,
Data Structures, and Program Development

1 Essential information

- This assignment is due Tuesday September 20th at noon and will be turned in on gradescope
- The assignment is intended to be done in pairs, although you're allowed to work on your own if you want.
- You can start the assignment before your lab period, although I recommend not getting too far – the lab TAs often have useful hints, and I don't want you wasting too much time struggling with something that the TAs might announce in-class.
- For more rules see the lab rules document on canvas.

2 Introduction

This laboratory is intended as a short introduction to python for students who are already familiar with a C-style language such as C, C++, or Java. The Python syntax in this lab might go *a little* beyond what's been covered in lecture formally, but should be achievable none-the-less. If you find yourself getting stuck on the python syntax – remember that this is normal.

You might find it useful to begin by planning out your program in pseudocode or a familiar programming language. Having a solid plan in this way will let you focus on ONLY the translating-into-python part of this lab. Remember, it's OK to ask the TAs for help with the python syntax, or to reference zybook or your notes for that.

Educationally, by doing this lab you should:

1. Set up (and test) your python programming environment
2. Practice the process of submitting code in this course
3. Make sure you have access to slack (so you can ask questions, attend office hours, and look at pet pictures and memes posted by the class)
4. Write your first Python functions.

Before outlining the assignment itself – you should start by setting up your python programming environment.

3 Software environment setup

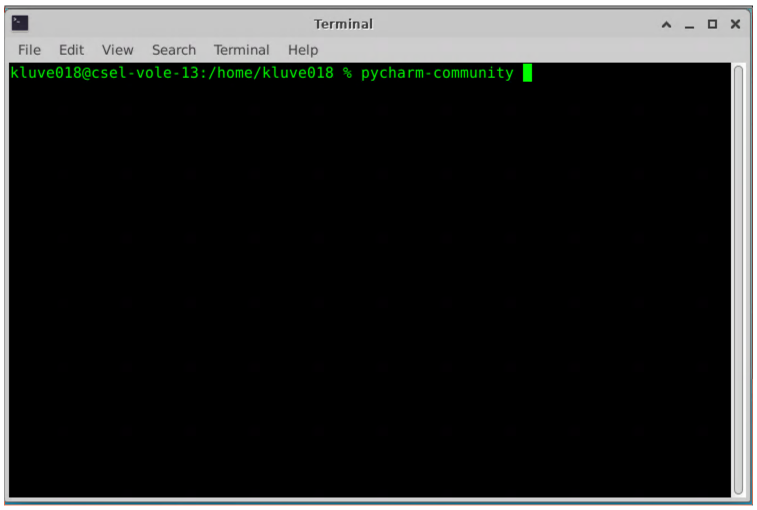
For the next few labs you will be doing python programming. Ultimately, all that matters is that you've created sufficient and correct python files. Therefore, you are free to use whichever text editor and programming environment you choose. That said, the recommended lab IDE (Integrated Development Environment) for the python part of this course is the PyCharm IDE. This section describes how to launch pycharm (on the CSELabs machines), and how to set up a new project for this lab.

3.1 Launching PyCharm

note This is where I would like to be able to confidently give you the rundown on the exact steps for launching pycharm for the first time on the CSELabs machines, but unfortunately, my (Daniel's) CSELabs account is no longer a reasonable reflection of yours (any given student's), and in past semesters we've learned that what works for me doesn't always work for students for launching programs. As such, my specific instructions may not always work. For this, it will be quite important to look to your lab staff for specific instructions, and possibly a quick demo.

It's possible that pycharm will have a good "launcher" setup in the applications menu. If so, just go to applications (top left corner), then the development submenu, and finally "PyCharm Community Edition". If that works, skip ahead to the project setup instructions.

If Pycharm doesn't seem have a launcher script set up in the application menu on CSELabs machines, then you will need to launch pycharm from the terminal. However, as this doesn't seem to be set up, you can just launch pycharm from the terminal with the command `pycharm-community`



```
kluve018@csel-vole-13:/home/kluve018 % pycharm-community
```

Depending on your past use of PyCharm you may need to enter some one-time settings such as color theme and default set of keyboard shortcuts. You can set these options however you want, while I prefer default key-bindings and a light theme, many people may disagree.

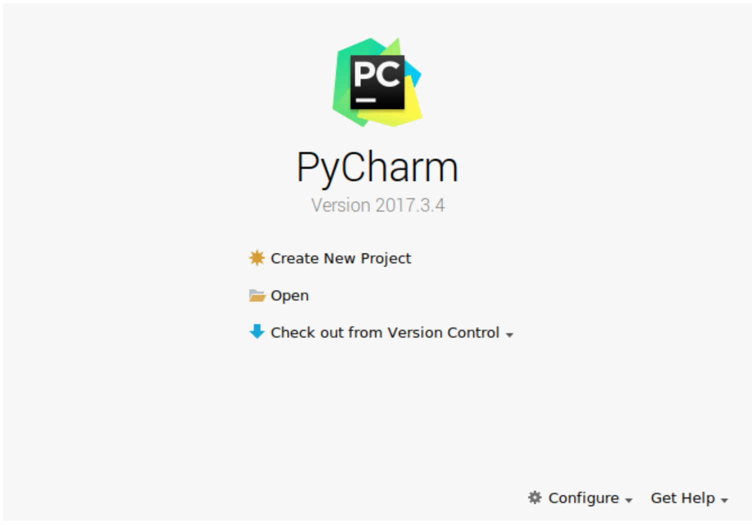
Ultimately, these decisions have a negligible effect on your coding output, and can be changed later from the settings menu. If you are not prompted for these settings, that too is fine.

3.2 Setting up a Project

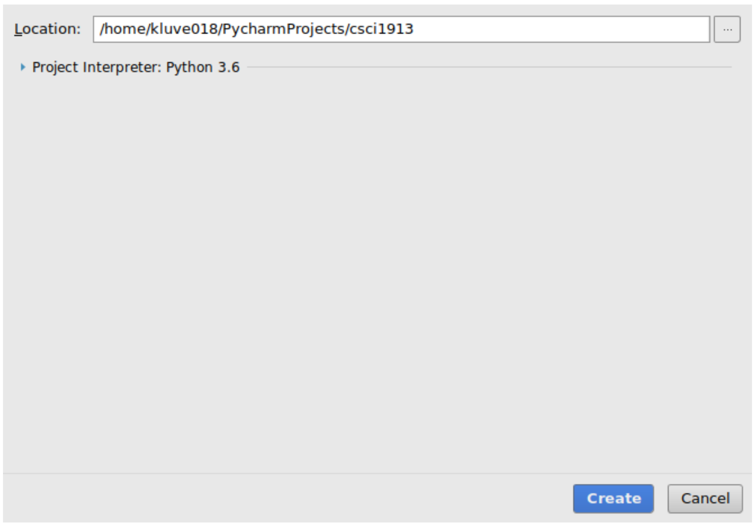
Many IDEs are organized around the concept of a "project" – a group of related code and file resources. It is rare for a programmer to work on a program that only has one file, often any interesting programming task would be split across many files, and may possibly have other associated resources (data files, configuration, etc.) These related files are, together, referred to as a project.

For this class, while labs often will be small, I would still recommend making a different Project for each lab. By keeping the labs separate you reduce the chance of any issues caused by reused variable, function, or file names. It is quite common for a programmer to have many projects on their computer (although it is rare for them to have more than one open at a time).

Once open, depending on settings and past use PyCharm will either open to it's splash-screen or a previously opened project. Either way you want to **make a new project**. If you are in a previous project go to File → new → Project. Otherwise you can simply click the "Create New Project" option in the splash screen:



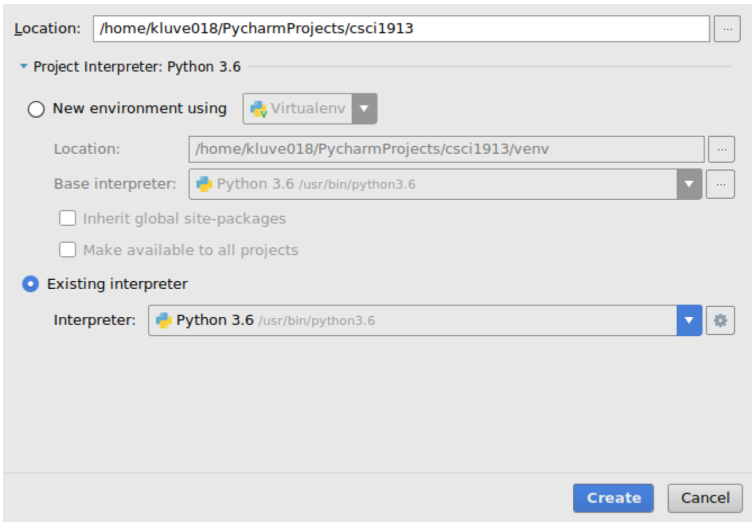
The new project window should look something like this:


Location: /home/kluve018/PycharmProjects/csci1913

Project Interpreter: Python 3.6

You can freely change the project Location, the default is fine too. Simply make sure you are confident you can find that location in your file browser. **It is very important to make a note of the project location.** You will need to find these files outside of the IDE to hand them in for grading.

You will also want to check your project interpreter by clicking the little triangle next to the project interpreter menu option. This setting effects several aspects of how PyCharm set's your project up. While most options are fine, the default option has a few extra files we don't need for our purposes. Switch your project interpreter to an existing interpreter, and make sure it's a python 3 interpreter (not a python 2 interpreter). (Note here, these screenshots are a little out-of-date, your python interpreter would probably be 3.8 not 3.6)


Location: /home/kluve018/PycharmProjects/csci1913

Project Interpreter: Python 3.6

New environment using Virtualenv

Location: /home/kluve018/PycharmProjects/csci1913/venv

Base interpreter: Python 3.6 /usr/bin/python3.6

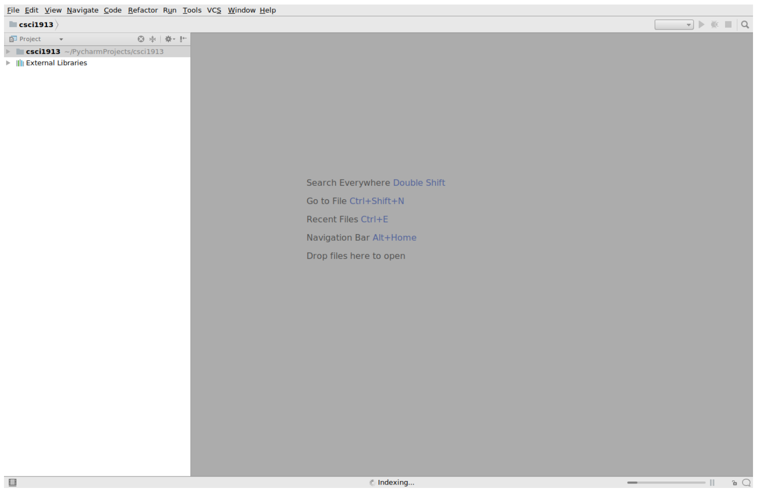
Inherit global site-packages

Make available to all projects

Existing interpreter

Interpreter: Python 3.6 /usr/bin/python3.6

Once you have that setting you can click next. This should bring you to the main project view itself.



This screen has a few parts, which you will get used to as you use it more. Most importantly, however, is the sidebar on the left, which gives you an overview of project files, and the main pane, in which your files will show up. You will do at least one meaningful project in this IDE, so it will be worth spending some time getting used to as you work, instead of just trying to push past it as fast as possible.

If the side bar doesn't show up, you will want to summon it with the appropriate keyboard command. On windows and Linux, under default keybindings, the keyboard command is alt + 1. If that doesn't work you can also toggle the side bar from the menu → View → Tool Windows → Project.

On the sidebar there should be a folder icon whose label is based on the project's name (I.E. I named mine csci1913, so the folder is labeled csci 1913, if you named yours lab1, it would be named lab1) That folder represents your main project folder in the file system. If you need to add a file to PyCharm you can simply click and drag the file from your file browser to that folder icon and PyCharm will copy that file to the right location. Likewise, to add a new file you can right-click on that icon and go to new → Python file. To view the contents of that folder you may need to click the little triangle next to the folder icon.

4 Python syntax notes

It's looking like we might not cover everything I hoped before labs, so I've included some syntax notes here that should cover everything needed to get over the line.

4.1 function declarations

The python syntax for declaring a functions is as follows:

```
def <name>(<parameters>):
    code
    code
    code
```

with parameters being simply a list of names separated by commas (like python variables, python parameters have no type) As an example:

```
def isDivisible(x, y):
    """A function to check if x is divisible by y"""
    return x % y == 0
```

4.2 Loops

The python for loop's basic syntax is as follows:

```
for <variable> in range(<limit>):
    code using variable
```

This loop will run `limit` times, with `variable` taking the values $0, 1, 2, \ldots, limit - 1$

Sometimes you want to loop from 1 up to (and including) limit. If you want to do that I recommend the following loop:

```
for <variable> in range(1,<limit>+1):
    code using variable
```

This loop will run `limit` times, with `variable` taking the values $1, 1, 2, \ldots, limit$

5 Introduction

The focus of this lab is the idea of a "Highly Composite Number" – a number which is divisible by more other numbers than any smaller number. This definition is *very mathematical* so it can be hard to really understand these numbers, and why they might be interesting. Let's build up to that by first reviewing divisors.

Simply put, the number of divisors an integer has is the number of integers that divide it (including 1 and itself). So for 9, we have three divisors (1, 3, and 9) and for 12 we have 6 divisors (1,2,3,4,6,12) The following table shows the number of divisors for each number up to 20

number	1	2	3	4	5	6	7	8	9	10	11	12	12	14	15	16	17	18	19	20
divisors	1	2	2	3	2	4	2	4	3	4	2	6	2	4	4	5	2	6	2	6

Computing the divisor count by-hand is simple enough for small numbers, but for a large number like 78,612,768 it would take a while to list all 120 divisors. This makes divisor counting an obvious choice for computer automation – while it would take forever to solve this problem, it would take only a second for a computer to check all 78,612,768 possible divisors and count which values divide without remainder.

The divisor count of a number can tell us a lot about the number, for example, numbers with very few divisors (I.E. numbers with 2 divisors) are prime numbers. Prime numbers are themselves a subject of a fair amount of study for their rare property of having few divisors. In this lab, however, we are interested in numbers with the opposite property – I.E. those with very many divisors.

The number 60 is a good example of a very divisible number, being divisible by 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60 (that's 12 different values). This is more divisors than any number smaller than it, and makes 60 well-suited when humans may need to perform division with 60. This might explain why 60 is a good choice for the number of minutes in an hour: 60 can be divided evenly by many different numbers, half-an-hour(30) a quarter-of-an-hour(15), a third-of-an-hour(20), and a tenth-of-an-hour(6) are all simple-to-compute integer numbers. If we used 100 minutes per hour, for example, a third of an hour would be 33 and one third minute – much harder to work with.

This brings us to our key definition for the lab: an *highly composite number* is formally defined as a positive integer with more divisors than any other smaller positive integer. The Wikipedia page on this topic explores this idea in more depth, but numbers like, 12, 24, 36, 60, 120, 1260 are all examples. Looking at the table above we can see that there are 5 highly composite numbers under 20: 1, 2, 4, 6, and 12. Each of these numbers has more divisors than all smaller numbers. Note that 16 is not highly composite: while it is the first number to have 5 divisors, 12 has more divisors with 6. While it might be quite tedious for a human to establish if a number is highly composite or not, it's rather easy for a computer. A computer can solve this with a "brute-force" algorithm (put simply, count how many divisors every smaller integer has and check if it is, indeed, bigger than all smaller integers) We will explore this approach in Lab 1.

6 Formal Requirements

You are required to create a python file named `compositeNumbers.py` which contains three function definitions:

1. num_divisors – this should count how many divisors a positive integer has
2. is_highly_composite_number – this should return True or False to indicate if a number is a highly composite number
3. kluver_cat_name – this function should return a string containing the name of my pet cat. I will post about my cat on Slack you can get the name there.

(You can have more functions if you want, just make sure you have all three of these!)

To make getting started on this easier I've included a base file `compositeNumbers_template_rename.py` on canvas. Your first step should be to download this file (and the tester file), rename it to the correct filename, and fill in your names in the comment at the top of the file. As you do this be careful – copy/pasting text from a PDF often includes formatting characters or spaces or other things that can make the text incorrect (yes – even if you copied it directly, it can still be wrong). Therefore, I recommend typing the filename manually. Remember – filenames are generally case-sensitive.

Once you've renamed the template file your next job is to run the tester. It will not output the correct values (the correct values can be seen in the test file itself) but it should run without crashing. **Do not begin programming until verifying that the tester is setup correctly to allow you to run your code and test it!** Knowing how to test your code on an ongoing basis is a vital skill, and the tests are a vital tool to allow you to check your code for obvious/major issues. Unfortunately, no testing tool can ever prove your code correct, but using these tools can help you catch most bugs in your code before grading.

6.1 num_divisors(n)

This function should take a single integer (n) and return an integer to indicate the number of divisors n has. The easiest way to do this is simple counting – check each number between 1 and n and count how many divide n evenly.

- If given a negative number or 0, the number 0 should be returned.
- The number 1, and the number n itself should both be counted as possible divisors
- A few examples that can help:
  - the number 1 has 1 divisor (itself)
  - the number 2 and 3 both have two divisors (1 and themselves).
  - The number 4 has 3 divisors (1, 2, 4) and so forth.
- Your function is not expected to handle non-integer data. We will not test your functions with non-integer data, and it's OK if your function crashes or otherwise "misbehaves" when given non-integer data.

Two hints here:

- remember that the modulus operator `%` can be used to compute the remainder of integer division – this might not sound useful, but you can check this remainder to see if two numbers are divisible
- zybooks chapter 2.15 shows the syntax for the `range` function, and the example at the end has a reminder how to use it with a for loop, should you need a reminder. Getting the start AND stop of your for loop right is important here (since you don't want to check 0 for being a divisor, but you do want to check n itself)

6.2 is_highly_composite_number(n)

This function should take a single integer (n) and return a boolean value (`True` or `False`) to indicate if the number fits the formal definition of highly composite: A number is highly composite if it is a positive integer with more divisors than any smaller positive integer.

- If given a negative number or 0 the function should return False (only positive numbers are highly composite)
- A number is only highly composite if it has strictly MORE divisors than all smaller positive integers.
- you are **required** to use your `num_divisors` function inside this function. If you don't know how to do this just ask and we'll show you.

- Your function is not expected to handle non-integer data. We will not test your functions with non-integer data, and it's OK if your function crashes or otherwise "misbehaves" when given non-integer data.

A few hints here:

- `True` and `False` are Boolean values, not strings – you can just type True or False to refer to them.
- It's possible to write this function without EVER typing True or False (hint – comparisons like `>` or `<` directly produce Boolean values.)
- Like any function – make sure **YOU** understand what this function needs to do and can perform the computation directly (if slowly). If you don't *fully* understand what's being asked you will **never** be able to solve this problem intentionally. There is nothing wrong with asking for clarifications or extra explanations on what's written in the PDF itself.
- The Wikipedia page for Highly Composite Numbers has a list of many examples you can use for testing, and their divisors. If you're struggling to understand this – that might help. **warning:** Your function must work for ALL highly composite numbers – not only the ones on the list on Wikipedia.
- This function can be split up into a few steps – and can be easier to think about if you do. Remember – planning is part of programming.

6.3 kluver_cat_name()

This function should take no input and return only a string. The string returned should contain the (case-sensitive) name of professor Kluver's cat. You should be able to find this information on slack without too much trouble. If you can't find it – ask the TAs for **help finding this information** (the TAs will not just tell you what my cat is named)

I'm aware this requirement might sound petty, vain, and a bit random, but there's actually two reasons behind it. The first purpose of this function is to make sure everyone has access to slack, and to get everyone to poke around slack, just a little bit. While there are rarely any access issues with slack I wanted to get those issues addressed as soon as possible (as slack is REQUIRED for online office hours, and is a key Q&A resource in this course). By putting required information from this assignment on slack we can make sure everyone checks their access to this important resource immediately, and that we are pro-active about resolving issues. I've seen student wait to access slack in the past, and they've uniformly told me "I wish I knew I could ask questions here sooner!" If you have trouble accessing slack – check the "sign up" link provided in both the syllabus and the course technology document on canvas, or reach out for assistance over email.

You will be in charge of writing the entire function declaration for this function. (This is actually the second reason for this function – practice the function syntax with a trivial function-body) Remember – this also includes a docstring – your code will run without the docstring, but it's considered bad code style and bad manners. More importantly, one of the tests formally checks if you have a docstring, and missing this will ALSO lose you points in the code style part of manual evaluation.

7 Grading

Grading on this assignment will be 50% automated tests, and 50% manual checks. The automatic grading will be broken down roughly as follows:

- 6 points for turning the file in, having the right file name, the right function names, and a file that loads.
- 18 points for num_divisors – this follows the compositeNumbersTester file found on canvas
- 18 points for is_highly_composite_number – this follows the compositeNumbersTester file found on canvas
- 8 kluver_cat_name – this function should return a string containing the name of my pet cat. I will post about my cat on Slack you can get the name there.

The Manual grading will be 10% code style, and 40% spot-checks / partial credit. The spot-checking will focus on

- Problems in how students approached coding (I.E. things you can do to pass tests without actually solving the problem – these strategies are, of course, not acceptable)
- Major code style issues (unreadable code, problematic code duplication, not using your num_divisors function to compute the number of divisors in your is_highly_composite_number function, etc.)
- code issues that make your code fail the test. You will get partial credit for this, and if the problem was a "small" one, you will also get some hints about how to avoid the issue in the future.
- Actual autograder mistakes (These are rare, but it's nice to double-check every now and again – of course if your code is right and the autograder is wrong we will refund the points.)

For code style, we will use the following style guidelines for this lab. Future labs may have a more restrictive style guideline (as will most workplaces).

- Every function should have a docstring
- Every file should have a comment at the top, which should say, at a minimum, your name and your partners name.
- Variable names should be as descriptive as possible (clearly `n` and `i` will be accepted here as common loop variables or math variables, but `c` instead of `count` would be considered a bad variable name choice)
- Reasonable use of whitespace – don't spread the lines of a function out with massive whitespace, don't use 1-space-only for indentation. That sort of thing.
- If there are problems that actively make your code hard to read, we reserve the right to mention them, even if they are not listed here. Ultimately, code is about communication, if your code doesn't make sense it's not correct (no matter how well it runs!)

8 Submission

Before submitting your work:

- Please use the tester code on canvas, and any tests you devise on your own, to test the code and carefully check that your code is bug-free. Buggy code is worth very little in the programming community – especially if the bugs prevent it from running.
- Make sure the file is named `compositeNumbers.py` (case sensitive)
- Make sure the that you name, and your partner's name, are in a comment at the top of the file
- If you and your partner are submitting one copy of the assignment – make sure whoever submits it marks their partner **in gradescope itself**.

You are only required to submit the `compositeNumbers.py` file, which will be submitted through gradescope. You can submit as many times as you want, only the final submission will be considered for grading.

If you worked with a lab partner, gradescope has a way to add your partner to your submission after you submit (to show yourself as having worked in a group). We prefer you use this feature if at all possible. Not using it may lead to trouble getting credit for your work. More information on this feature can be found here:
`https://help.gradescope.com/article/m5qz2xsnjy-student-add-group-members` If you worked with a lab partner, please make sure both you and your partner have a copy of the lab files for future reference.

If you finish this lab during the lab time, feel free to notify your lab TAs, and then leave early. If you do not finish this lab during the lab time you are responsible for finishing and submitting this assignment before the deadline.