

CS-119 Lab #4

Expected Learning Objectives

- Creating a Python console application
- Declaring variables
- Assigning values to variables
- Arithmetic operations Python repetition (loop) structures.

Overview

This lab provides an opportunity to apply the concepts and principles presented in PLD chapter 4.

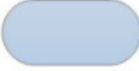

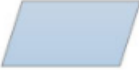
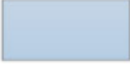

Exercise 1 – Guided Activity

Loops or repetition structures are used extensively in software development so it's very important to get comfortable coding these. The exercise we're going to do here isn't necessarily real practical but it does illustrate the use of loops for a couple of common arithmetic operations. The first one we'll do is an exponent function. Most modern programming languages have either an exponent operator (I.e., Visual Basic, Python) or a built-in library function (Java, C#) to calculate exponents so, once again, we're doing this just to get some hands-on practice coding loops. The other arithmetic operation we'll do is the factorial function.

This time around, we'll do a flowchart. Keep in mind a flowchart is just like pseudocode except in a more graphical form.

Creating a Flowchart

Quick review of the basic flowchart shapes:

Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectangle represents a process
	Decision	A diamond indicates a decision

For doing exponents, we know from past math classes the expression 2^4 is really a shorthand notation for the expression $2*2*2*2$. This looks like something we could do with a loop, right? Let's give it a shot. We'll use the online tool [Draw.io](https://draw.io). It functions very similar to Microsoft Visio.

Start out by creating a new diagram.

← → ↻ https://www.draw.io



File Edit View Arrange Extras Help



100%



Device

Create New Diagram

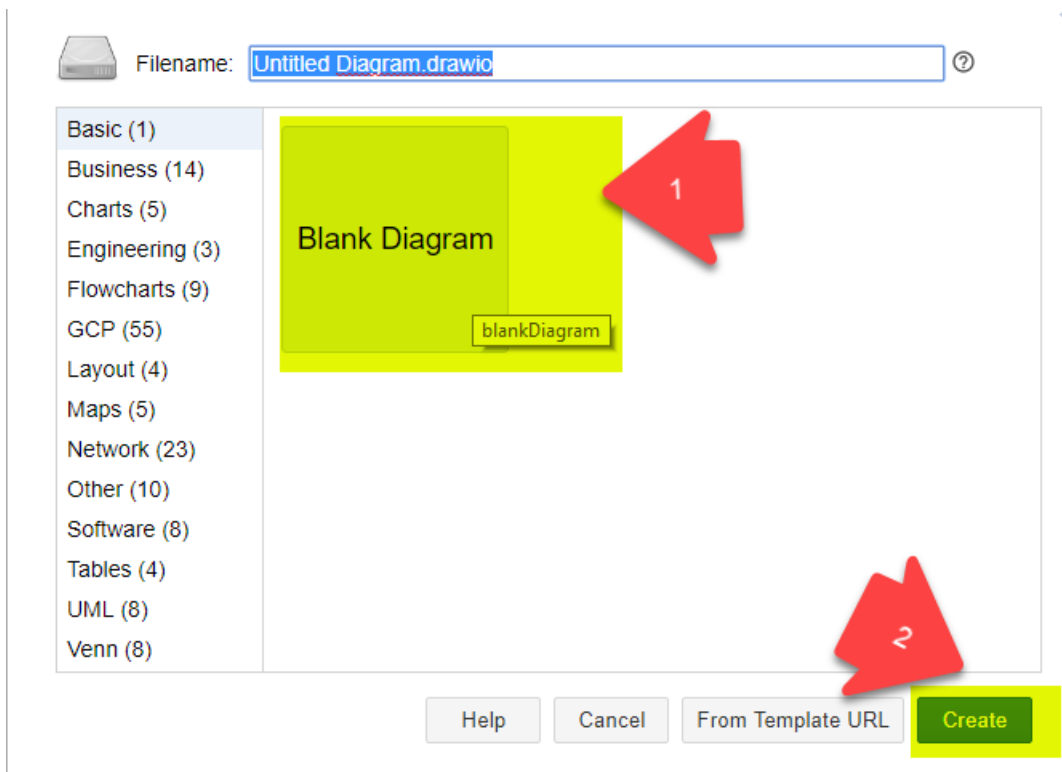
Open Existing Diagram

[Change storage](#)

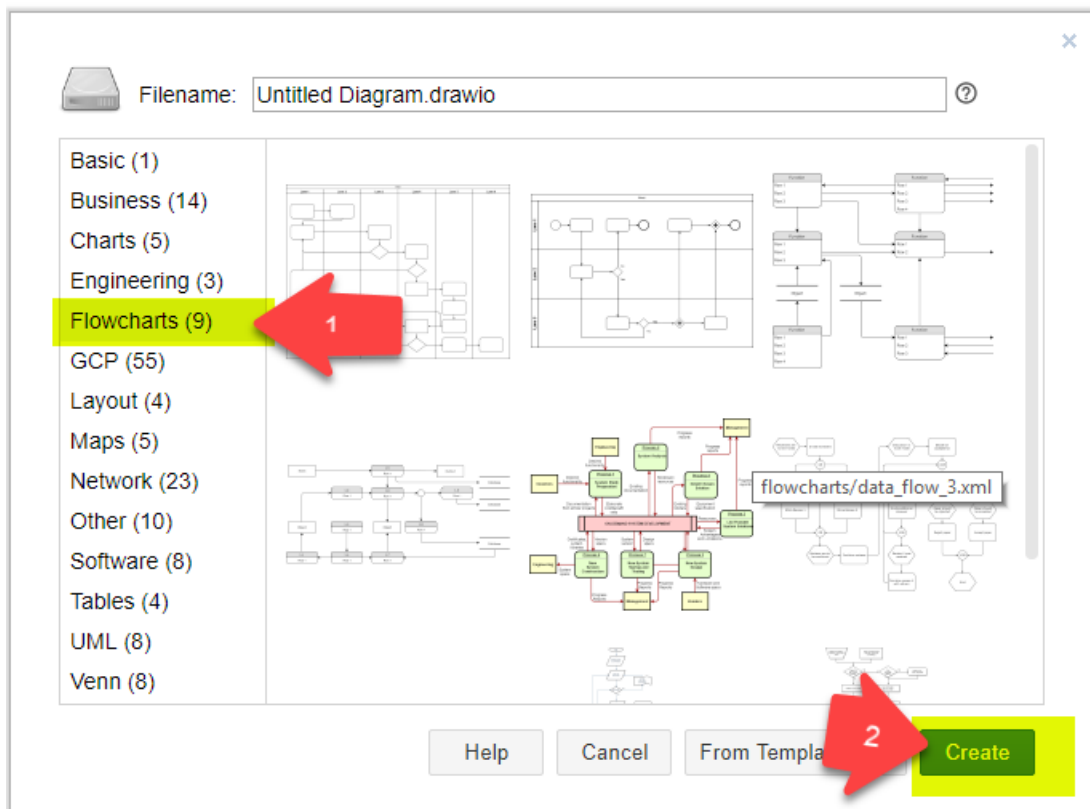
Help

Language

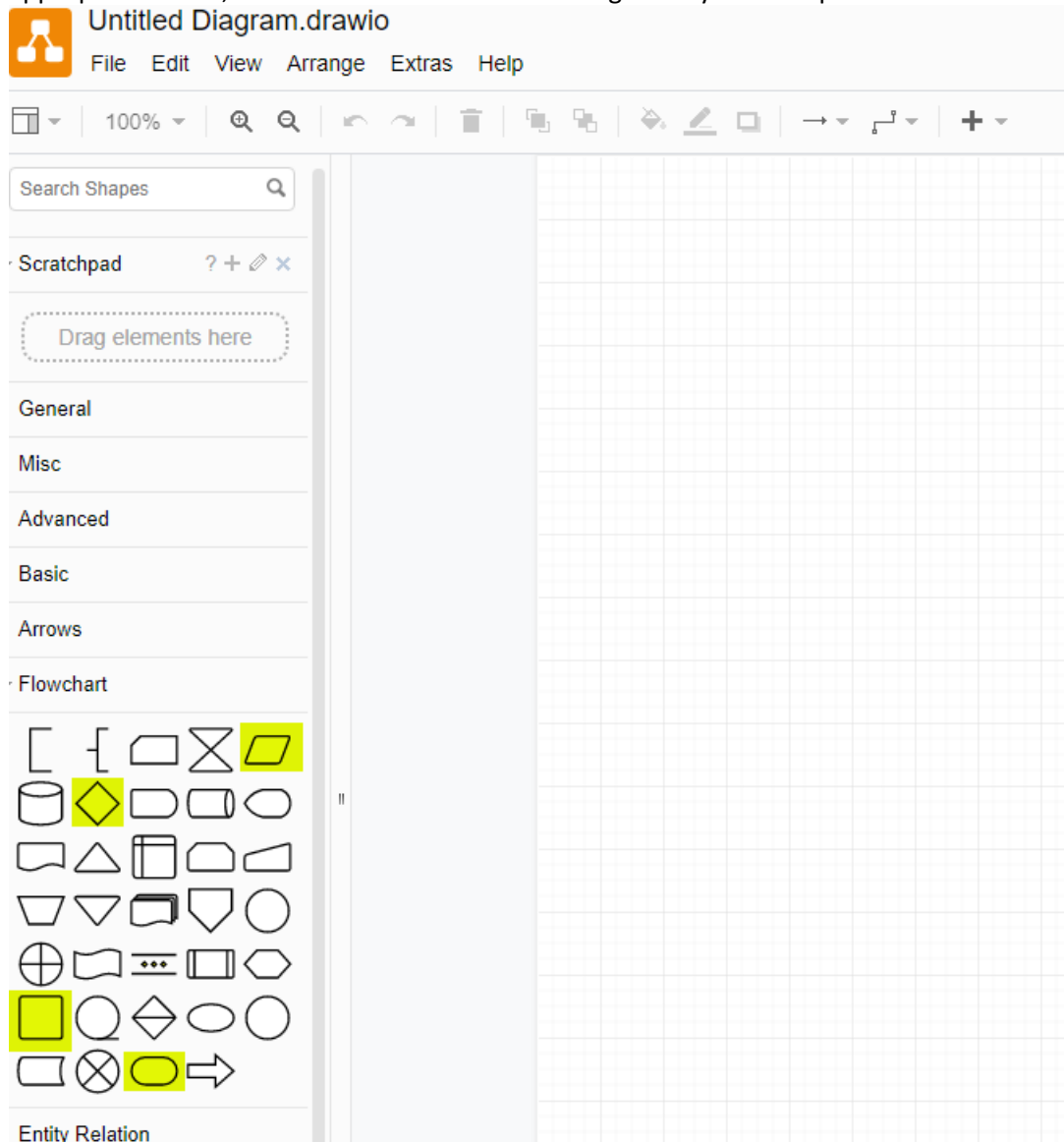
Select (1) Blank Diagram and then (2) Create.



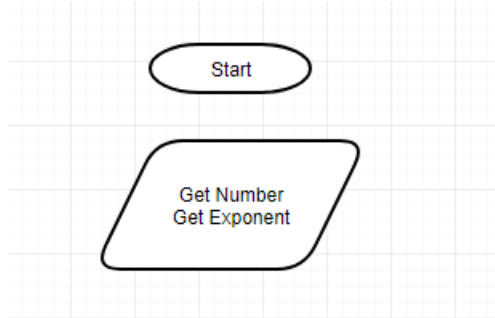
You can also select Flowcharts and then Create.



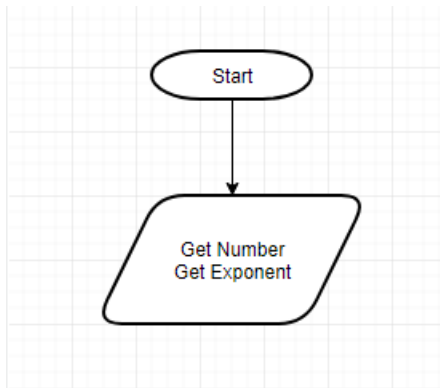
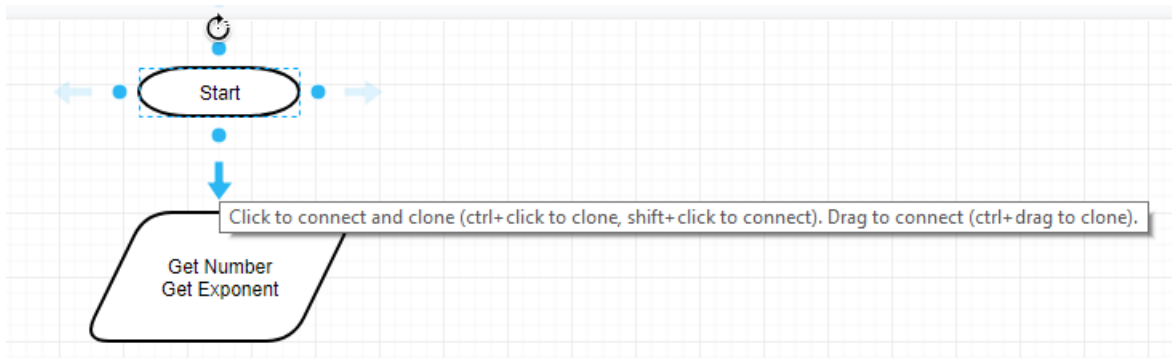
You will get a blank drawing with various tool palettes of shapes on the left side. Expand Flowchart. The “key” shapes we’ll be using are highlighted below. Select a shape, hold down the left mouse button, and then drag and then drop the shape onto the page. Double-click the shape to add text. A “proper” flowchart will have a distinct start and end; all shapes will have appropriate labels, and all structures will have a single entry and exit point.



First, drag and drop a Start/End oval (or “lozenge” as the textbook likes to call it) onto the page. Double-click the oval and label it as either “Start” or “Begin”. Drag and drop a Data (Input/Output) parallelogram onto the page. To save time and space, we’ll combine our inputs into one shape. Double-click the parallelogram and enter the required inputs. In this case, we need the number and the power or exponent. Your flowchart should look something like this:



Click on the Start oval and drag the flow line/arrow to the parallelogram. You can also use the Connector tool in the tool bar to draw flow lines. Be aware getting flowlines drawn correctly can be a bit of a trial and error process. The flowchart should now look something like this:

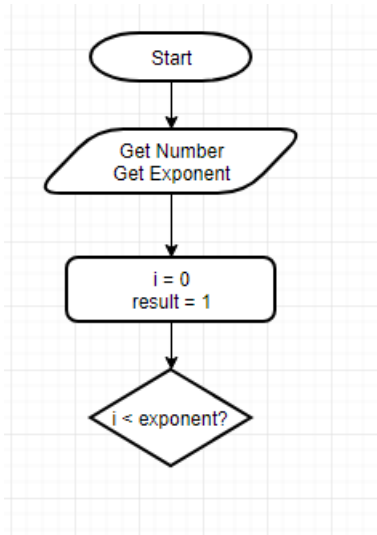


Note how the arrow on the connector always points to the next step.

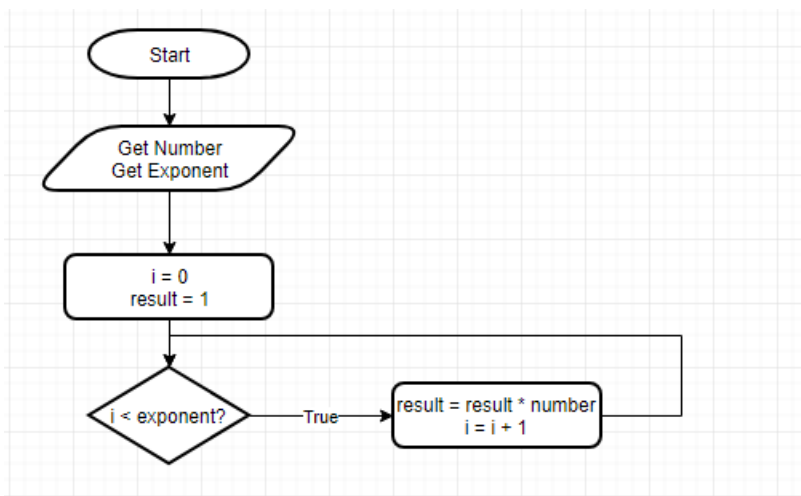
Next, drag and drop a process box (rectangle) onto the page, double-click and initialize the loop counter with the statement $i = 0$. Also, we need a variable for the result that will be initialized to 1 so add the statement $result = 1$. When coding loops, it’s generally accepted and understood the single letter variables i , j and k are loop counters. If you prefer to use a variable name such

as *count*, that's fine too. Draw a line from the input to the process box. The flow chart should now look like this:

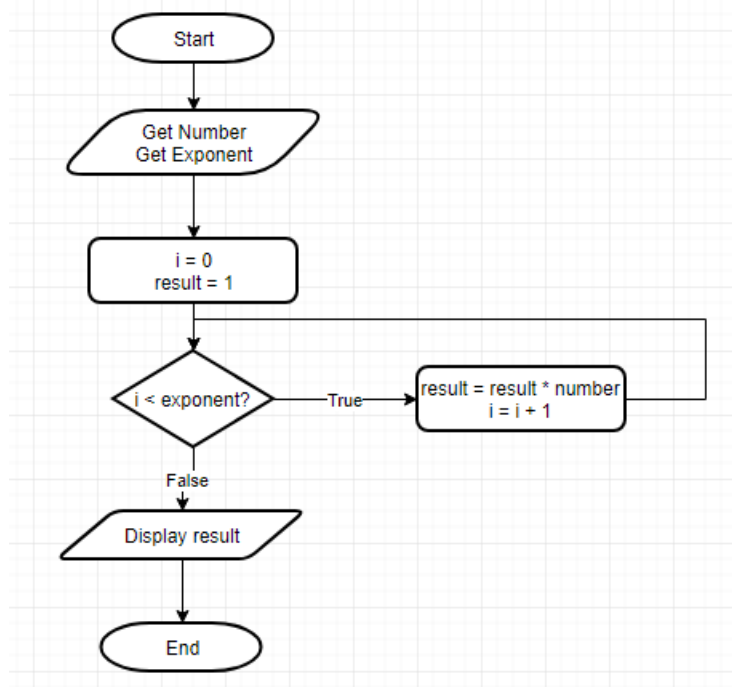
Next, add a decision diamond to test the loop condition. Since we initialized $i = 0$, we need to ask the question $i < \text{exponent}$. Draw a flow line into the top of the decision diamond. The flowchart will now look like this:



In a decision diamond, there will always be 2 flow lines coming out; one for the true path and 1 for the false path. For a loop, when the loop condition is false, the loop will terminate. We'll arbitrarily start with the true path so we need to add a process box to set $\text{result} = \text{result} * \text{number}$ and, very important, increment the loop counter by 1. Again, to save time and space, we'll put both in the same box. Draw a flow line from the right side of the decision diamond to the process box. Double-click the flow line and label *true*. It's also generally accepted to use Yes/No, T/F or Y/N Instead of True/False. Add a flow line from the process box to the top of the decision diamond. Our flowchart should now look something like this:



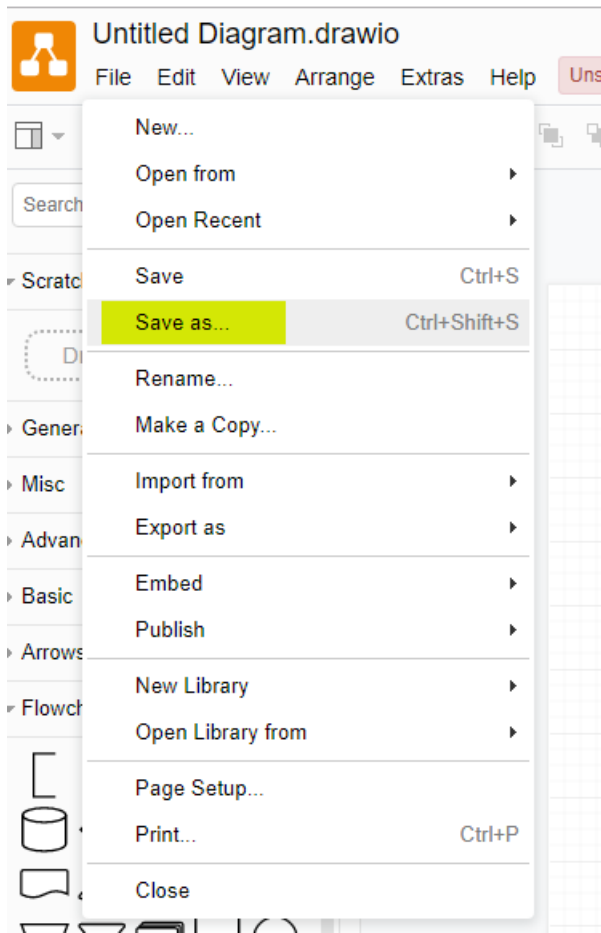
When the loop condition is false, the loop will terminate and the next step will be to add another data (input/output) parallelogram to display the result. Add a flowline labeled false from the bottom of the decision diamond to our output step. Last, add a terminator and a flowline from the output. Our finished flowchart will look like this:



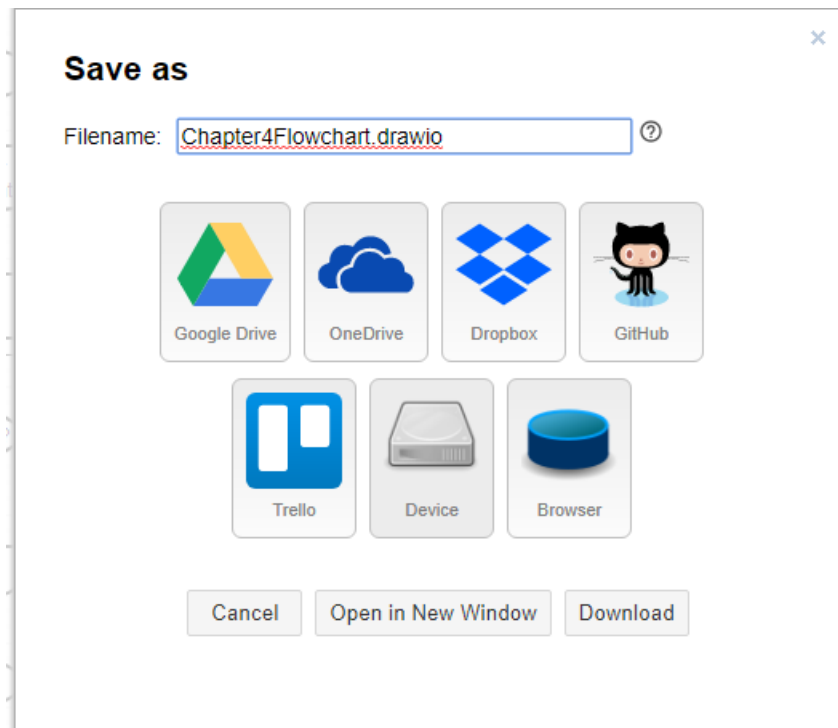
Things to notice about a “properly drawn” flowchart:

- Distinct start/end points.
- Flowlines run from top to bottom, left to right with the exception of flow lines going to the top of a loop.
- Flowlines in a decision are clearly labeled with the true and false path.
- Arrows on flow lines point to the next step in the process.
- Each program structure has exactly one entry and one exit point; no steps lead to a dead end.

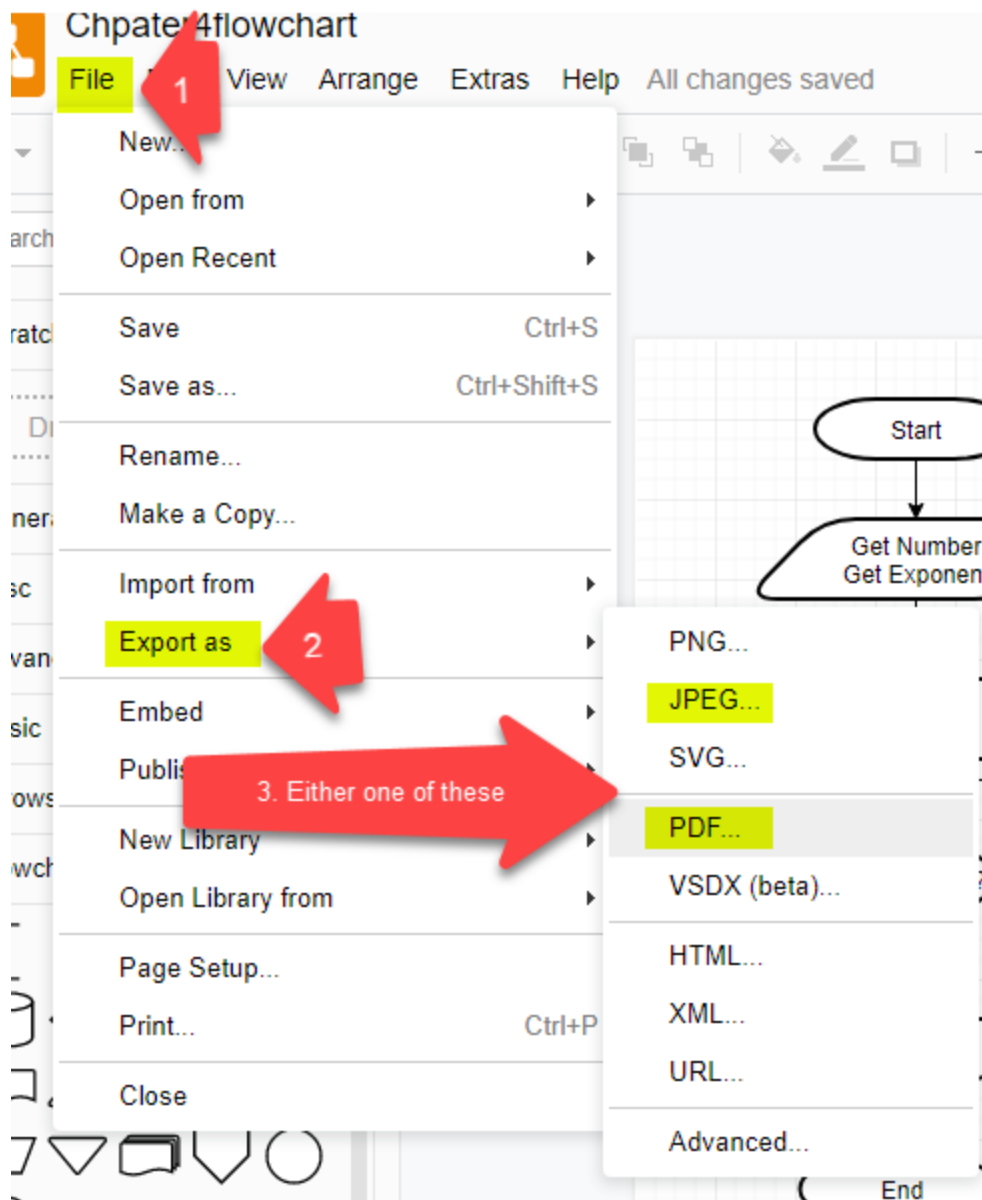
Last thing: Be sure to save your work! If you plan to edit or do work on your flowchart later on, be sure to click on File and then “Save as”



Enter a suitable file name. If you want to save to your computer, select “Device”. You may also save to one of the online cloud based accounts shown below assuming you have a user account. By default, Draw.io saves to the Windows user downloads directory. Be sure to copy or move the file to your lab folder.



For turning in a lab, export to either PDF or JPEG format. Be sure the file is in your lab folder.



For comparison purposes, we could write pseudocode something like this:

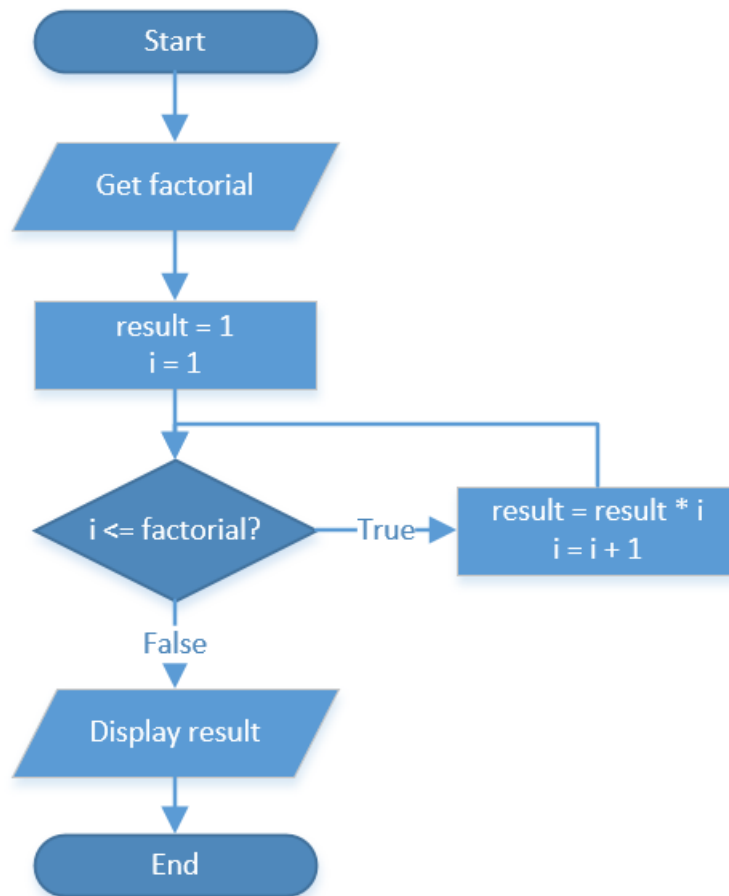
```
Get number
Get exponent
i = 0
result = 1

while i < exponent
    result = result * number
    i = i + 1
end while

Display result
```

For the factorial exercise, recall from a math class somewhere along the way that the expression $5!$ is simply a short hand notation for $1*2*3*4*5$. Once again, does this look like something we could code using a loop?

Refer to the previous exercise if you need details on how to create a flowchart, add shapes, draw flow lines, etc. The factorial flowchart should look something like this:



For both exercises, notice how we initialized our result variable to 1 because multiplying by 0 will give a result of 0. Be aware! Little things like this are very common logical errors in programming!

To save time and to make things easier, you may do both flowcharts on the same page.

Python Implementation

Create a file called Exponent.py. For those of you using lab computers, be sure to save to your USB drive or H: drive and be sure to save to your lab 3 folder. This is worth points!

The next step is to convert our pseudocode into real working Python code.

```
#!/user/bin/env python3

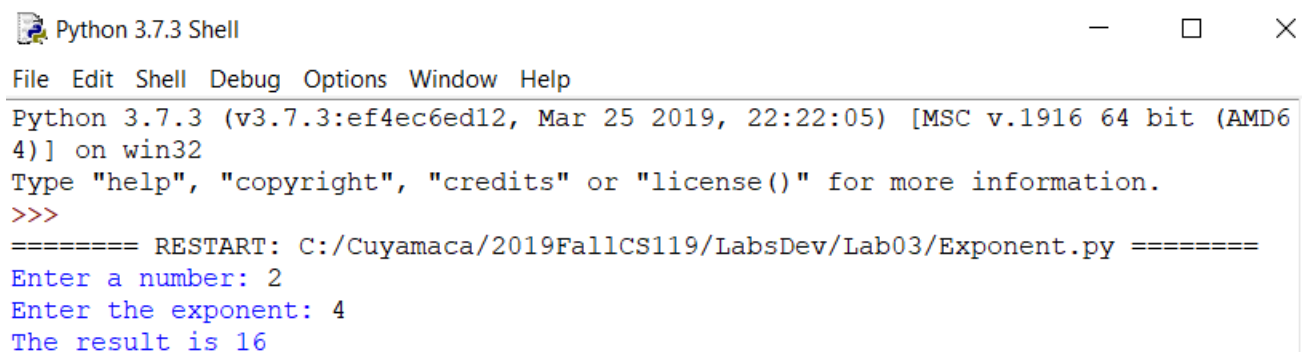
# Cuyamaca College CS-119
# While loop demo definite loop with a fixed number of iterations

# variables and constants
number = 0
exponent = 0

# input - get number and exponent
number = int(input("Enter a number: "))
exponent = int(input("Enter the exponent: "))
i = 0
result = 1

# Make sure the loop body code is indented 4 spaces!
# Also, note the colon (:)
# The Ctrl c key combination will break you out of an endless loop
# Print to the console
while i < exponent:
    result = result * number
    i += 1 # Be sure to change the loop control variable!

print("The result is " + str(result))
```



The screenshot shows a window titled "Python 3.7.3 Shell" with standard window controls. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The status bar at the bottom indicates "Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)] on win32". The main text area shows the following output:

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Cuyamaca/2019FallCS119/LabsDev/Lab03/Exponent.py =====
Enter a number: 2
Enter the exponent: 4
The result is 16
```

Next, we'll do the factorial code. Create a new Python code file called Factorial.py. Enter the code below:

```
#!/user/bin/env python3

# Cuyamaca College CS-119
# John Gerstenberg
# Lab 4 Exercise 1 Factorial

# variables
factorial = 0
i = 1
result = 1

# input: get factorial
factorial = int( input("Enter a number: ") )

# just for fun, we'll do this with a for loop
for i in range(1, factorial + 1, 1):
    result = result * i

# output - display result
print("The result is " + str(result) )
```

Python 3.7.3 Shell

File Edit Shell Debug Options Window Help

Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>

===== RESTART: C:\Cuyamaca\2019FallCS119\Labs\Lab04\Exponent.py =====

Enter a number: 2
Enter the exponent: 4
The result is 16
>>>

===== RESTART: C:\Cuyamaca\2019FallCS119\Labs\Lab04\Factorial.py =====

Enter a number: 5
The result is 120
>>>

Ln: 12 Col: 4

Note the test values above to ensure each is working correctly.

Exercise 2

Create a Python application that will allow the user to calculate the future value of an investment. The user will enter a monthly amount, annual interest rate and number of years for the investment.

You will code a loop to calculate the future value and display the value of each month and the final value.

Some key formulas and free Python code snippets:

Note that the term is in years but we need to show the value of each month so a 5 year savings account would be 60 months.
 $\text{months} = \text{years} * 12$

Interest rates are a percentage and entered as 3.25 (i.e., for 3.25%), 1.125, etc. This is also an annual percentage rate (APR). We need to divide this rate by 100 and then by 12 to get the monthly percentage rate.

$\text{monthlyIntRate} = \text{annualIntRate} / 100 / 12$

The futureValue variable is an *accumulator* as it accumulates value as the loop executes.

$\text{futureValue} = (\text{futureValue} + \text{monthlyAmount}) * (1 + \text{monthlyRate})$

Steps

1. You can create either a flowchart or pseudocode to document your design. This is worth points and please complete this *before* attempting to write code!
2. Start Python IDE. Create a new file called Lab3FutureValue in your \Labs\Lab3 folder.
3. Code the application. Some of the code is given. The parts that are blurred out you get to code!


```
#!/user/bin/env python3

# Cuyamaca College CS-119
# Lab 3 future value

# variables and constants
monthly_amt = 0.0
years = 0
annual_int_rate = 0.0
monthly_int_rate = 0.0
fv = 0.0
months = 0
month_num = 1

# message with Python format specifiers
msg = "Month: {0:3d} FV: ${1:.2f} Interest: ${2:.2f}"
interest = 0.0
tot_interest = 0.0

# input - get monthly amt, period and interest rate
monthly_amt = float(input("Enter monthly amount: "))
annual_int_rate = float(input("Enter annual interest rate: "))
years = int(input("Enter number of years: "))

# process
monthly_int_rate = annual_int_rate / 12
months = years * 12

while month_num <= months:

    # format the message here and plug in the variable values
    print(msg.format(month_num, fv, interest))
    month_num += 1 # Be sure to change the loop control variable!

print("\n\nTotal interest: ${0:.2f}".format(tot_interest))
```

4. Test your application using the following values: Monthly Amount = 100; Interest Rate = 6; Years = 5; Value at 60 months should be somewhere around \$7011.88. Results should be *reasonably* close (+/- 2%). Don't worry if your results have a large number of decimal places. Number formatting is *optional* for this exercise. If you like a little extra challenge and want to add it in, there is some code in the chapter 3 lab that shows how to do this.
5. **Challenge option (optional):** Calculate the total interest earned. After the loop, add a line to display the total interest earned.

Exercise 3

In this exercise, you will get some experience with the random number generator in Python. Random numbers are heavily used in many games and simulations for things like dice rolls, dealing cards, flipping a coin, etc. You will build an application that will flip a coin 20 times, keep a running total of heads and tails. For each coin toss, you will display the result (heads or tails). After the last toss, you will display the total number of heads, total number of tails and the percentage of each.

Steps

1. Complete a flowchart *or* you may use pseudocode to develop your logic. This is worth points and please complete this before attempting to write code!
2. Create a new Python file called HeadsTails.
3. Code the application. Be sure to import random. A screen shot of a partially coded application is given here to get you started:

```
#!/user/bin/env python3

# Cuyamaca College CS-119
# Lab 3 coin flip

import random

# variables and constants
MAX_FLIPS = 20
MAX_VAL = 2
heads = 0.0
tails = 0.0
pct_heads = 0.0
pct_tails = 0.0
i = 0
flip = 0

for i in range(1, MAX_FLIPS + 1, 1):
    flip = random.randint(0,1)

print("Total heads: {}".format(heads))

pct_heads = (heads / MAX_FLIPS) * 100.0

print("Pct heads: {:.1f}".format(pct_heads))
```

4. Test your application.

Grading Criteria:

Deliverable	Points	Breakdown
Exercise 1 guided flowchart	10	Completed for both the exponent and factorial. Proper use of symbols and flow lines.
Exercise 1 guided code	5	Complete, code "makes sense", results are correct.
Exercise 2 flowchart or pseudocode	5	Complete, "makes sense". Input, processing and output clearly defined. If flowchart, proper use of symbols and flow lines.
Exercise 2 code and run	15	Complete, "makes sense". Appropriate variable names, constants, comments, code is easily readable. Compiles and produces correct results
Exercise 3 flowchart or pseudocode	5	Complete, "makes sense". Input, processing and output clearly defined. If flowchart, proper use of symbols and flow lines.
Exercise 3 code and run GUI	10	Complete, "makes sense". Appropriate variable names, constants, comments, code is easily readable. Compiles and produces correct results. All required outputs are displayed
Lab Total	50	