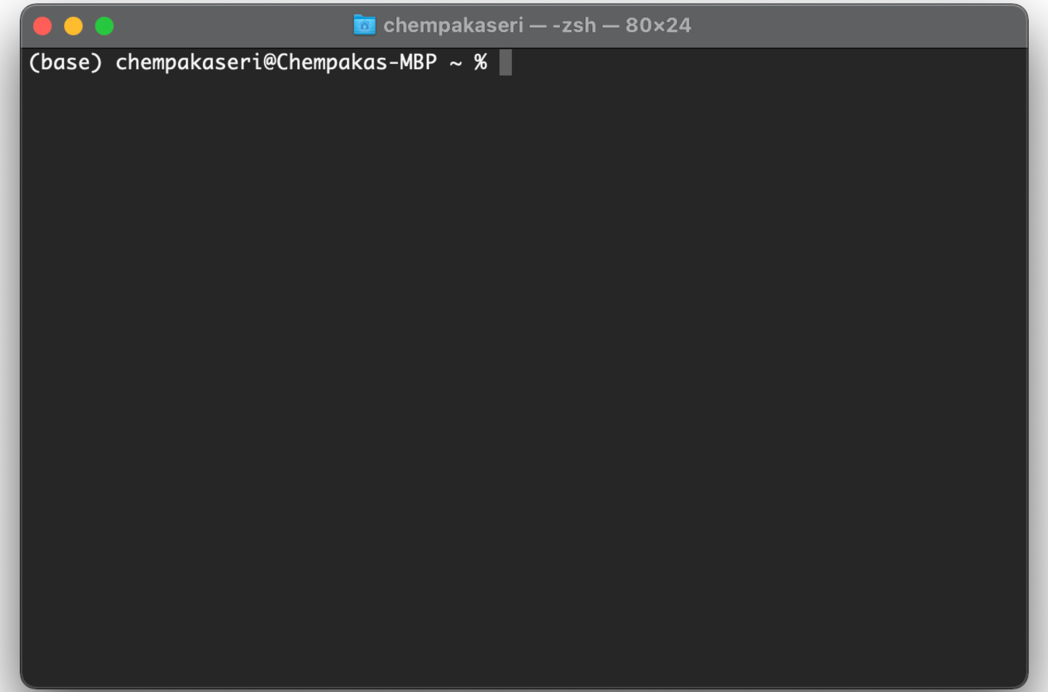
Building Python Programs

Chapter 3: Parameters and Graphics

Conda Environment

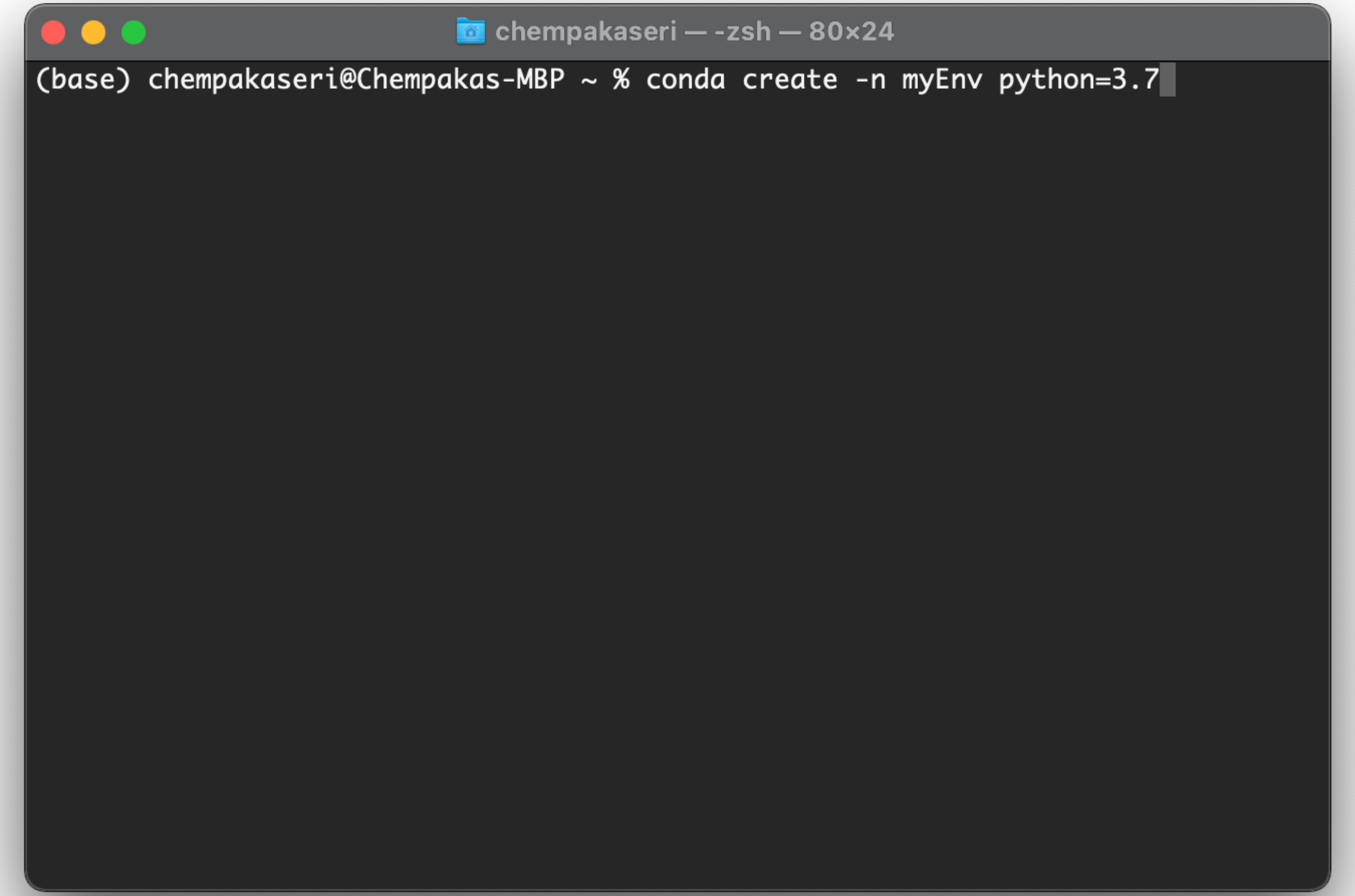
Conda Environment

- A conda environment is a directory that contains a specific collection of conda packages that you have installed
- A virtual environment is a tool that helps to keep dependencies required by different projects separate by creating isolated spaces for them that contain per-project dependencies for them



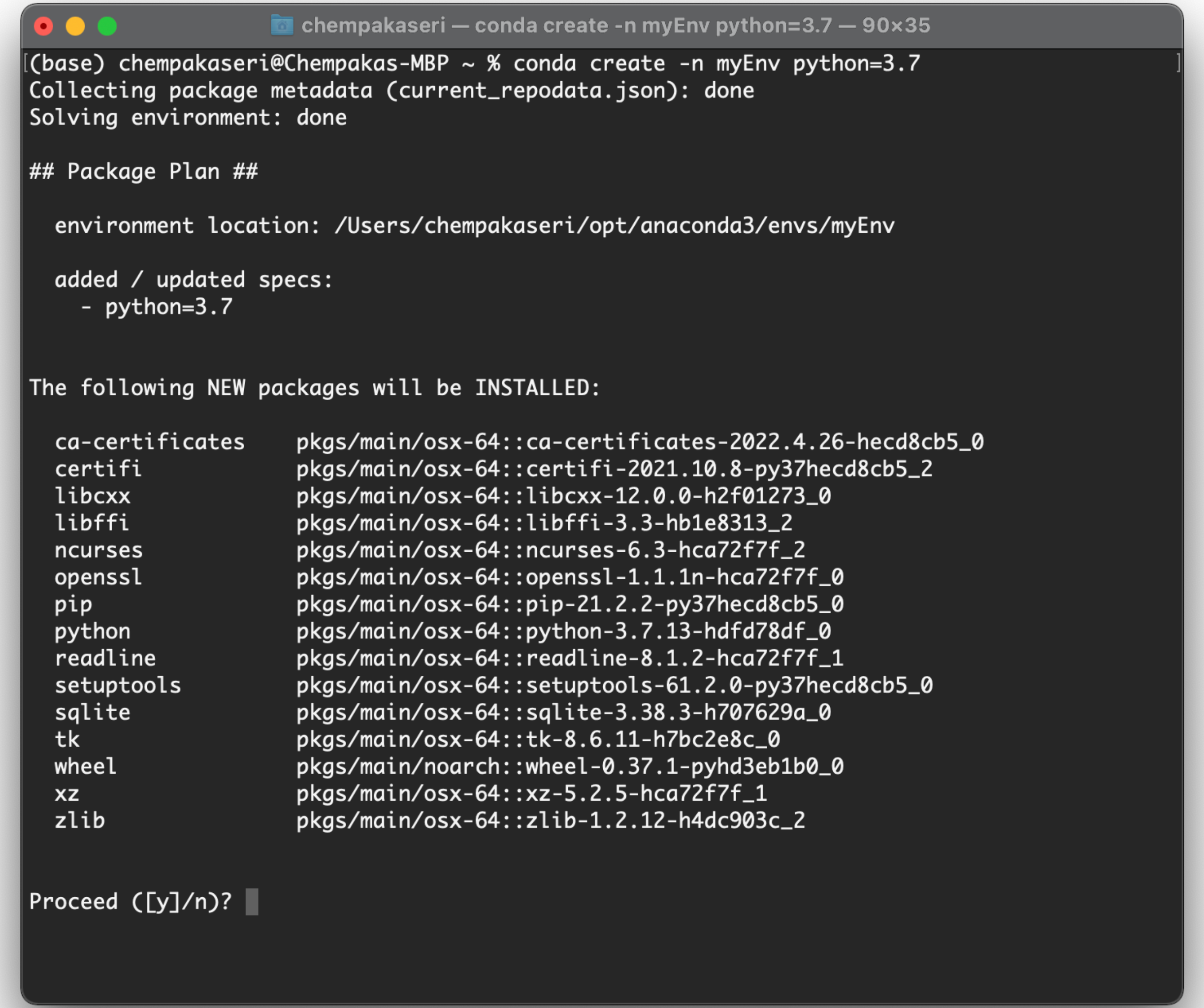
```
(base) chempakaseri@Chempakas-MBP ~ %
```

Create Environment



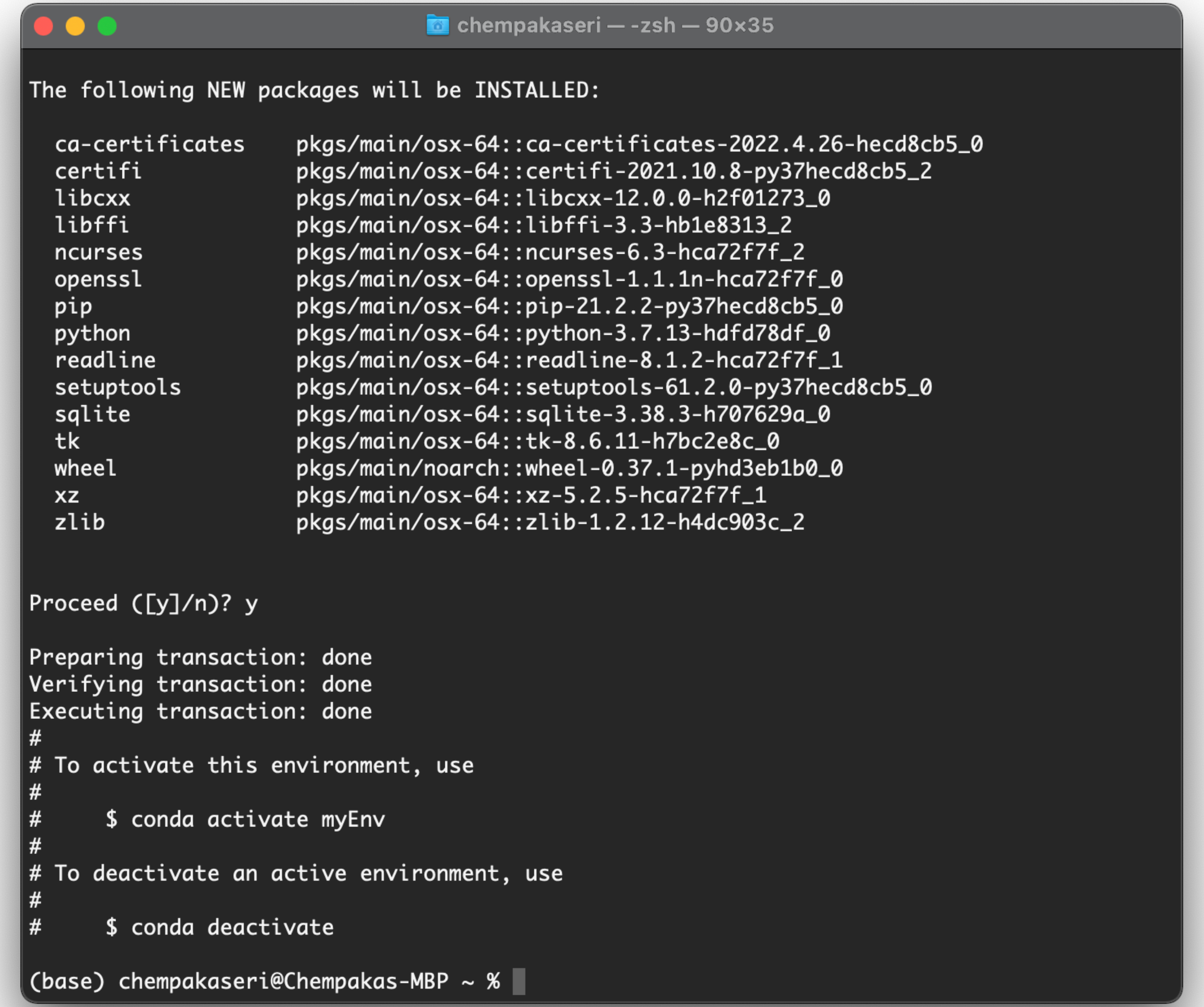
```
(base) chempakaseri@Chempakas-MBP ~ % conda create -n myEnv python=3.7
```

Create Environment



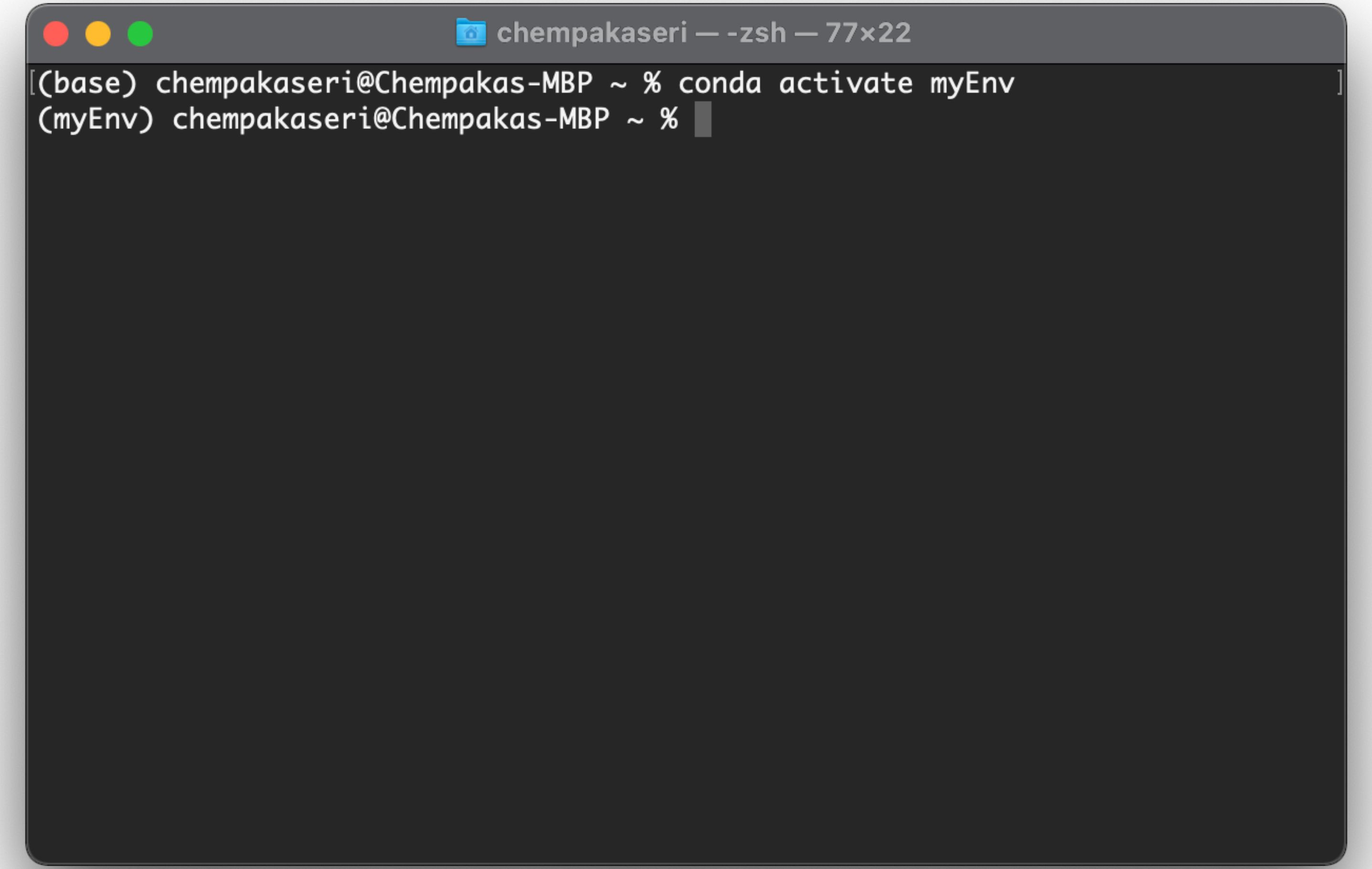
```
(base) chempakaseri@Chempakas-MBP ~ % conda create -n myEnv python=3.7
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /Users/chempakaseri/opt/anaconda3/envs/myEnv

  added / updated specs:
    - python=3.7


The following NEW packages will be INSTALLED:

  ca-certificates    pkgs/main/osx-64::ca-certificates-2022.4.26-hecd8cb5_0
  certifi            pkgs/main/osx-64::certifi-2021.10.8-py37hecd8cb5_2
  libcxx             pkgs/main/osx-64::libcxx-12.0.0-h2f01273_0
  libffi             pkgs/main/osx-64::libffi-3.3-hb1e8313_2
  ncurses            pkgs/main/osx-64::ncurses-6.3-hca72f7f_2
  openssl            pkgs/main/osx-64::openssl-1.1.1n-hca72f7f_0
  pip                pkgs/main/osx-64::pip-21.2.2-py37hecd8cb5_0
  python             pkgs/main/osx-64::python-3.7.13-hdfd78df_0
  readline           pkgs/main/osx-64::readline-8.1.2-hca72f7f_1
  setuptools         pkgs/main/osx-64::setuptools-61.2.0-py37hecd8cb5_0
  sqlite             pkgs/main/osx-64::sqlite-3.38.3-h707629a_0
  tk                 pkgs/main/osx-64::tk-8.6.11-h7bc2e8c_0
  wheel              pkgs/main/noarch::wheel-0.37.1-pyhd3eb1b0_0
  xz                 pkgs/main/osx-64::xz-5.2.5-hca72f7f_1
  zlib               pkgs/main/osx-64::zlib-1.2.12-h4dc903c_2


Proceed ([y]/n)?
```

Create Environment



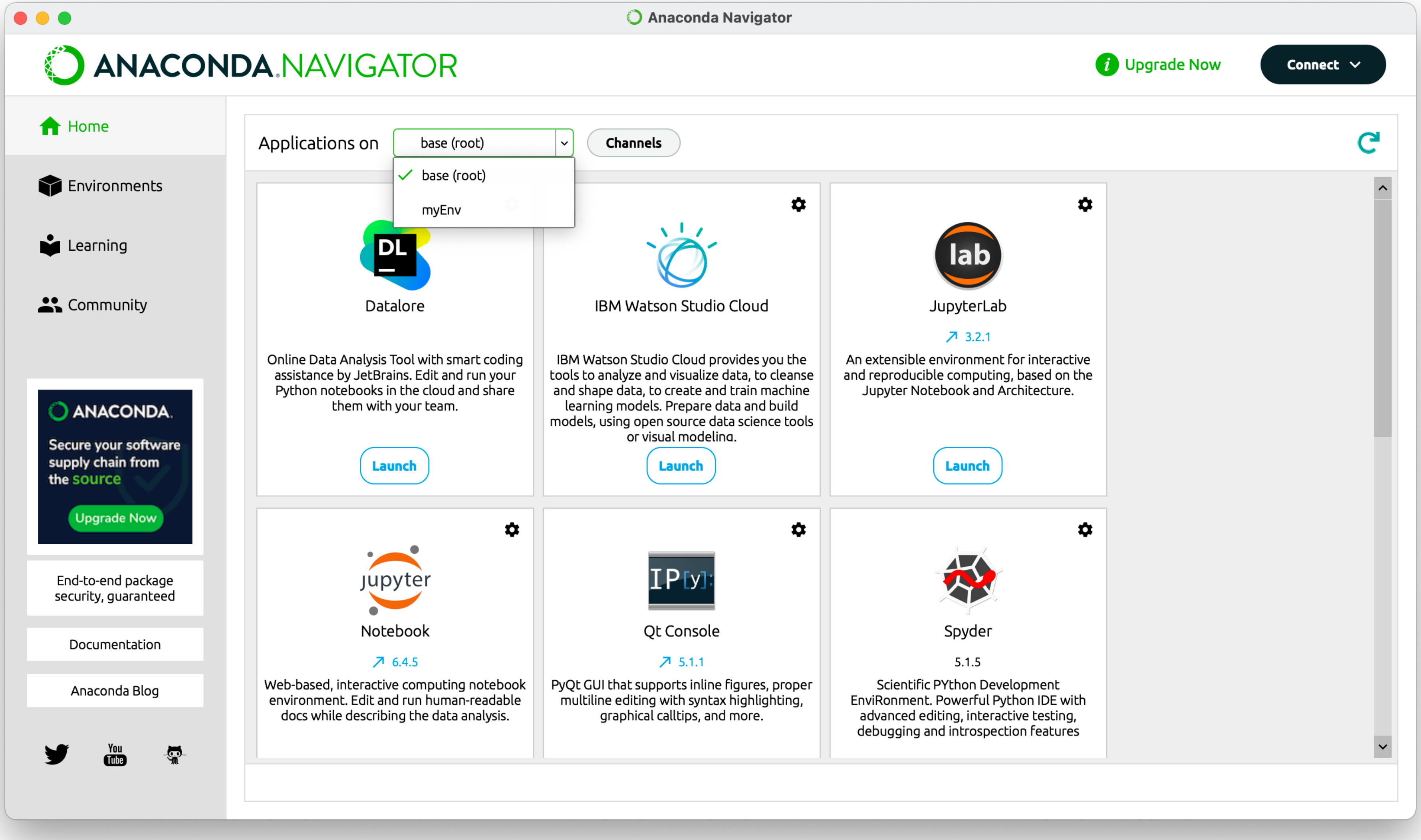
```
The following NEW packages will be INSTALLED:

  ca-certificates    pkgs/main/osx-64::ca-certificates-2022.4.26-hecd8cb5_0
  certifi            pkgs/main/osx-64::certifi-2021.10.8-py37hecd8cb5_2
  libcxx             pkgs/main/osx-64::libcxx-12.0.0-h2f01273_0
  libffi             pkgs/main/osx-64::libffi-3.3-hb1e8313_2
  ncurses            pkgs/main/osx-64::ncurses-6.3-hca72f7f_2
  openssl            pkgs/main/osx-64::openssl-1.1.1n-hca72f7f_0
  pip                pkgs/main/osx-64::pip-21.2.2-py37hecd8cb5_0
  python             pkgs/main/osx-64::python-3.7.13-hdfd78df_0
  readline           pkgs/main/osx-64::readline-8.1.2-hca72f7f_1
  setuptools         pkgs/main/osx-64::setuptools-61.2.0-py37hecd8cb5_0
  sqlite             pkgs/main/osx-64::sqlite-3.38.3-h707629a_0
  tk                 pkgs/main/osx-64::tk-8.6.11-h7bc2e8c_0
  wheel              pkgs/main/noarch::wheel-0.37.1-pyhd3eb1b0_0
  xz                 pkgs/main/osx-64::xz-5.2.5-hca72f7f_1
  zlib               pkgs/main/osx-64::zlib-1.2.12-h4dc903c_2


Proceed ([y]/n)? y

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate myEnv
#
# To deactivate an active environment, use
#
#     $ conda deactivate

(base) chempakaseri@Chempakas-MBP ~ %
```

Activate Environment



```
(base) chempakaseri@Chempakas-MBP ~ % conda activate myEnv
(myEnv) chempakaseri@Chempakas-MBP ~ % 
```

Activate Environment - Spyder



Datalore

Online Data Analysis Tool with smart coding assistance by JetBrains. Edit and run your Python notebooks in the cloud and share them with your team.

IBM Watson Studio Cloud

IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.

JupyterLab

↗ 3.2.1

An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook

↗ 6.4.5

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Qt Console

↗ 5.1.1

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Spyder

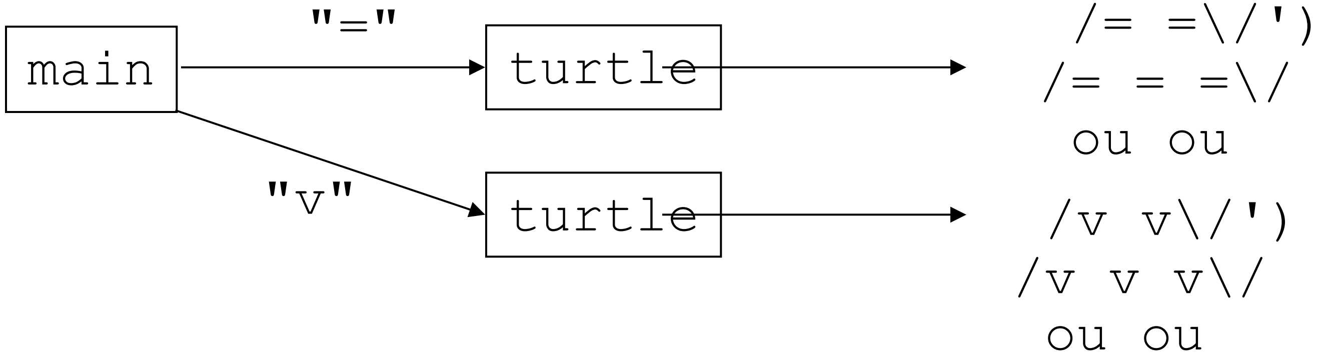
5.1.5

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

parameters

Parameterization

- **parameter**: A value passed to a function by its caller.
  - Instead of `turtle_equal`, `turtle_v`, write `turtle` to draw any turtle.
    - When *declaring* the function, we will state that it requires a parameter for the number of stars.
    - When *calling* the function, we will specify how many stars to draw.



```
 /= =\/')
/= = =\/
 ou ou

 /v v\/')
/v v v\/
 ou ou
```

Declaring a parameter

Stating that a function requires a parameter in order to run

```
def <name> (<name>):
    <statement>(s)
```

- Example:

```
def say_password(code):
    print("The password is:", code)
```

- When `say_password` is called, the caller must specify the code to print.

Passing a parameter

Calling a function and specifying values for its parameters

<name> (***<expression>***)

- Example:

```
say_password(42)
say_password(12345)
```

Output:

```
The password is 42
The password is 12345
```

Parameters and loops

- A parameter can guide the number of repetitions of a loop.

```
chant(3)


def chant(times):
    for i in range(0, times):
        print("Just a salad...")
```

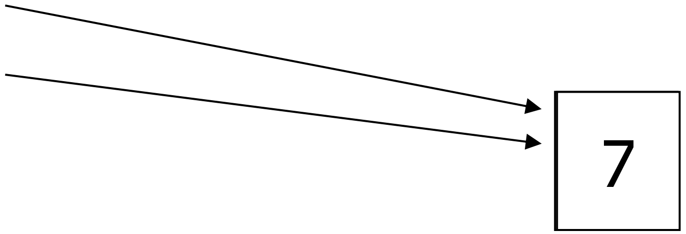
Output:

```
Just a salad...
Just a salad...
Just a salad...
```

How parameters are passed

- When the function is called:
  - The value is stored into the parameter variable.
  - The function's code executes using that value.

```
chant(3)
chant(7)
```



```
def chant(times):
    for i in range(0, times):
        print("Just a salad...")
```

Common errors

- If a function accepts a parameter, it is illegal to call it without passing any value for that parameter.

```
    chant()          # ERROR: parameter value required
```

- The value passed to a function must be of a type that will work.

```
    chant(3.7)       # ERROR: must be of type int if it
                    #          is used as a range bound
```

Multiple parameters

- A function can accept multiple parameters. (separate by ,)
  - When calling it, you must pass values for each parameter.

- Declaration:

```
def <name>(<name>, ..., <name>):
    <statement>(s)
```

- Call:

```
<name>(<exp>, <exp>, ..., <exp>)
```

Multiple parameters example

```
def main():
    print_number(4, 9)
    print_number(17, 6)
    print_number(8, 0)
    print_number(0, 8)

def print_number(number, count):
    for i in range(0, count):
        print(number, end="")
    print()
```

Output:

```
444444444
171717171717

00000000
```

A "Parameter Mystery" problem

```
def main():
    x = 9
    y = 2
    z = 5

    mystery(z, y, x)

    mystery(y, x, z)
```

□ □ □

```
def mystery(x, z, y):
    print(z, "and", (y - x))
```

Value semantics

- **value semantics**: When `numbers` and `strings` are passed as parameters, their values are copied.
  - Modifying the parameter will not affect the variable passed in.

```
def strange(x):
    x = x + 1
    print("1. x = ", x)

def main():
    x = 23
    strange(x)
    print("2. x = ", x)
...
```

Output:

```
1. x = 24
2. x = 23
```

returns

Python's Math class

Method name	Description
math.ceil(*value*)	rounds up
math.floor(*value*)	rounds down
math.log(*value, base*)	logarithm
math.sqrt(*value*)	square root

math.sinh(*value*)
math.cosh(*value*)
math.tanh(*value*)
	sine/cosine/tangent of an angle in radians

math.degrees(*value*)
math.radians(*value*)
	convert degrees to radians and back

Constant	Description
e	2.7182818...
pi	3.1415926...

`import math` necessary to use the above functions

Other math functions:

Function name	Description
abs(*value*)	absolute value
min(*value1, value2*)	smaller of two values
max(*value1, value2*)	larger of two values
round(*value*)	nearest whole number

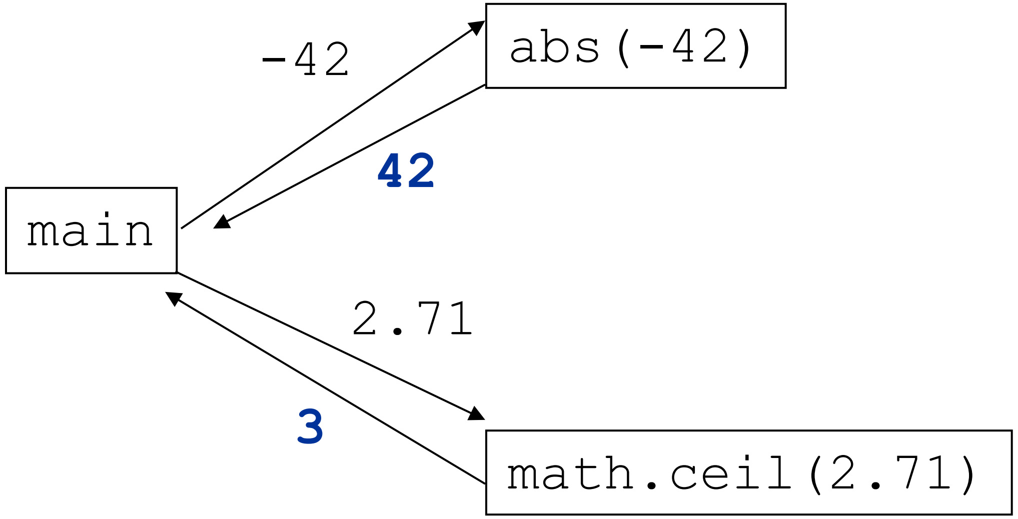
No output?

- Simply calling these functions produces no visible result.
  - `math.sqrt(81)    # no output`
- Math function calls use a Python feature called *return values* that cause them to be treated as expressions.
- The program runs the function, computes the answer, and then "replaces" the call with its computed result value.

  -
    ```
    math.sqrt(81)        # no output
     9.0               # no output
    ```
- To see the result, we must print it or store it in a variable.
  - `result = math.sqrt(81)`
  - `print(result)           # 9.0`

Return

- **return**: To send out a value as the result of a function.
  - Return values send information *out* from a function to its caller.
    - A call to the function can be used as part of an expression.
  - (Compare to parameters which send values *into* a function)



`Math` questions

- Evaluate the following expressions:
  - `abs(-1.23)`
  - `math.sqrt(121.0) - math.sqrt(256.0)`
  - `round(pi) + round(e)`
  - `math.ceil(6.022) + math.floor(15.9994)`
  - `abs(min(-3, -5))`

- `math.max` and math.`min` can be used to bound numbers.

  Consider a variable named `age`.
  - What statement would replace negative ages with 0?
  - What statement would cap the maximum age to 40?

Why return and not print?

- It might seem more useful for the `math` functions to print their results rather than returning them. Why don't they?

- Answer: Returning is more flexible than printing.
  - We can compute several things before printing:

    ```
    sqrt1 = math.sqrt(100)
    sqrt2 = math.sqrt(81)
    print("Powers are", sqrt1, "and", sqrt2)
    ```

  - We can combine the results of many computations:

    ```
    k = 13 * math.sqrt(49) + 5 - math.ceil(17.8)
    ```

Quirks of real numbers

- Some `float` values print poorly (too many digits).

```
result = 1.0 / 3.0
print(result)         # 0.3333333333333
```

- The computer represents `float`s in an imprecise way.

```
print(0.1 + 0.2)
```

  - Instead of 0.3, the output is `0.30000000000000004`

Type casting

- **type cast**: A conversion from one type to another.
  - To truncate a `double` from a real number to an integer

- Syntax:

  type (expression)

  Examples:

```
result = 19 / 5                 # 3.8
result2 = int(result)           # 3
x = int(sqrt(121))              # 1000
```

Returning a value

```
def name(parameters):
    statements
    ...
    return expression
```

- When Python reaches a return statement:
  - it evaluates the expression
  - it substitutes the return value in place of the call
  - it goes back to the caller and continues after the method call

Return examples

```
# Converts degrees Fahrenheit to Celsius.
def f_to_c(degrees_f):
    degrees_c = 5.0 / 9.0 * (degrees_f - 32)
    return degrees_c

# Computes triangle hypotenuse length given its side lengths.
def hypotenuse(a, b):
    c = math.sqrt(a * a + b * b)
    return c
```

- You can shorten the examples by returning an expression:

```
def f_to_c(degrees_f):
    return 5.0 / 9.0 * (degrees_f - 32)
```

Common error: Not storing

- Many students incorrectly think that a `return` statement sends a variable's name back to the calling method.

```
def main():
    slope(0, 0, 6, 3)
    print("The slope is", result);  # ERROR: cannot find symbol: result

def slope(x1, x2, y1, y2):
    dy = y2 - y1
    dx = x2 - x1
    result = dy / dx
    return result
```

Fixing the common error

- Returning sends the variable's *value* back. Store the returned value into a variable or use it in an expression.

```
def main():
    s = slope(0, 0, 6, 3)
    print("The slope is", s)

def slope(x1, x2, y1, y2):
    dy = y2 - y1
    dx = x2 - x1
    result = dy / dx
    return result
```

Exercise

- In physics, the *displacement* of a moving body represents its change in position over time while accelerating.
  - Given initial velocity v_0 in m/s, acceleration a in m/s^2, and elapsed time t in s, the displacement of the body is:
  - Displacement = $v_0 t + \frac{1}{2} a t^2$
- Write a method `displacement` that accepts v_0, a, and t and computes and returns the change in position.
  - example: `displacement(3.0, 4.0, 5.0)` returns `65.0`

Exercise solution

```
def displacement(v0, a, t):
    d = v0 * t + 0.5 * a * (t ** 2)
    return d
```

Interactive Programs

Interactive programs

interactive program: Reads input from the console.

- While the program runs, it asks the user to type input.
- The input typed by the user is stored in variables in the code.

- Can be tricky; users are unpredictable and misbehave.
- But interactive programs have more interesting behavior.

`input`

- **`input`**: An function that can read input from the user.

- Using an `input` object to read console input:

  name = `input(`**prompt**`)`

  - Example:

    ```
    name = input("type your name: ")
    ```

    - The variable `name` will store the value the user typed in

input example

```
def main():
    age = input("How old are you? ")

    years = 65 - age
    print(years, " years until retirement!")
```

age 29

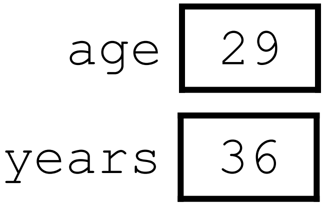
- Console (user input underlined):

```
How old are you? 29
```

```
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    print(65 - age)
TypeError: unsupported operand type(s) for -:
'int' and 'str'
```

input example

```
def main():
    age = int(input("How old are you? "))

    years = 65 - age
    print(years, "years until retirement!")
```


age 29

years 36

- Console (user input underlined):

```
How old are you? 29
36 years until retirement!
```

Random

Pseudo-Randomness

- Computers generate numbers in a predictable way using a mathematical formula

- Parameters may include current time, mouse position
  - In practice, hard to predict or replicate

- True randomness uses natural processes
  - Atmospheric noise (http://www.random.org/)
  - Lava lamps (patent #5732138)
  - Radioactive decay

Random

- `random` generates pseudo-random numbers.
  - `random` can be accessed by including the following statement:
    `import random`

Method name	**Description**
`random.random()`	returns a random float in the range [0, *1*) in other words, 0 inclusive to *1* exclusive
`random.randint(`***min, max***`)`	returns a random integer in the range [min, *max*] in other words, min to *max* inclusive

- Example:

```
import random
random_number = random.randint(1, 10)   # 1-9
```

Generating random numbers

- To get a number in arbitrary range [*min*, *max*] inclusive:

  `random.randint(`***min, max***`)`

  - Where ***size of range*** is (***max*** `-` ***min*** `+ 1`)

- Example: A random integer between 4 and 10 inclusive:

```
n = random.randint(4, 10)
```