Python Basics

And how to make the computer do the work for you

Quantitative Methods Workshop 2025
January 2nd 2025
10:00am – 12:00pm
Georgina Woo

The Plan!!

10am – 12pm

5m	Welcome!
15m	Programming tools
1h 15m	Core Python concepts
20m	Kahoot 1

1:45pm-4:45pm

45m	More Python concepts
15m	Kahoot 2
1h 45m	Programming time
15m	AMA/Review/Feedback

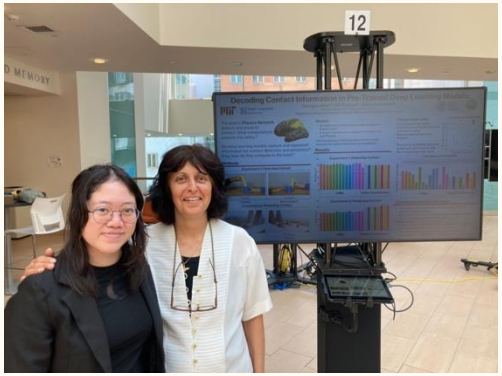
Theatre designer -> CS major @ Hunter College -> QMW 2024 -> MSRP 2024





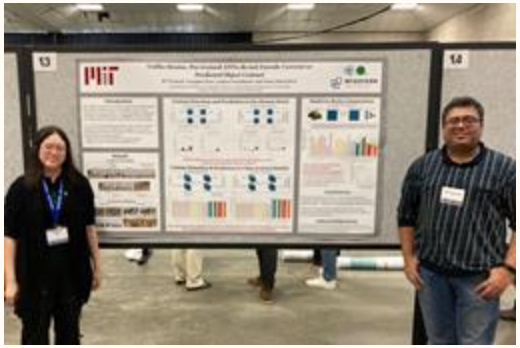

My Creatures

Kanlab!



Mandana with a fan

(Kanlab is still eating)



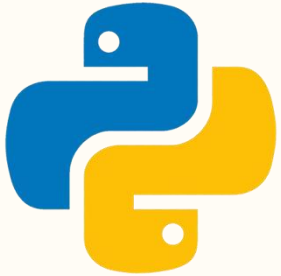
My MSRP supervisor Pramod

What is Python?

Python is a **high-level** programming language known for its readability and ease of use.

High Level

Highly abstract, human-readable code



Python

```
# Extract a substring using slicing
dna = "ATGCGTACG"
substring = dna[2:6]  # Extract characters from index 2 to 5
print(substring)  # Output: "GCGT"
```

High-ish Level

Still readable

```cpp
#include <iostream>
#include <string>

int main() {
    std::string dna = "ATGCGTACG";
    std::string substring = dna.substr(2, 4);  // Start at index 2, length 4
    std::cout << substring << std::endl;  // Output: "GCGT"

    return 0;
}
```

C++

Low Level

Noooo.

```
.data
dna:        .asciiz "ATGCGTACG"  # Null-terminated string
substring:  .space 5             # Reserve 4 characters + null terminator

.text
.globl main
main:
    la $t0, dna        # Load address of dna into $t0
    la $t1, substring  # Load address of substring into $t1
    addi $t0, $t0, 2   # Move to the starting index (2)

    li $t2, 4          # Number of characters to copy

copy_loop:
    lb $t3, 0($t0)     # Load a byte from dna
    sb $t3, 0($t1)     # Store the byte into substring
    addi $t0, $t0, 1   # Move to the next character in dna
    addi $t1, $t1, 1   # Move to the next position in substring
    subi $t2, $t2, 1   # Decrement the counter
    bgtz $t2, copy_loop  # If counter > 0, repeat

    sb $zero, 0($t1)   # Null-terminate the substring

    # Exit (MIPS syscall for exit)
    li $v0, 10
    syscall
```

MIPS

Under the hood

```
 1   0011 1100 0000 1000 0001 0000 0000 0000
 2   0011 1100 0000 1000 1000 0000 0000 0000
 3   0011 1100 0000 1001 0001 0000 0000 0000
 4   0011 1100 0000 1001 1000 0000 0000 1010
 5   0010 1000 1000 1000 0000 0000 0000 0010
 6   0010 1000 1010 0000 0000 0000 0000 0100
 7   1000 0000 0010 1000 0000 0000 0000 0000
 8   1010 0000 0011 1001 0000 0000 0000 0000
 9   0010 1000 1000 1000 0000 0000 0000 0001
10   0010 1000 1001 1001 0000 0000 0000 0001
11   0010 1000 1010 1010 1111 1111 1111 1111
12   0001 1100 1010 0000 1111 1111 1111 1010
13   1010 0000 0000 1001 0000 0000 0000 0000
14   0010 1000 0001 0000 0000 0000 0000 1010
15   0000 0000 0000 0000 0000 0000 0000 1100
```

Why Python?

Python is beginner friendly, has extensive library support, and is widely used in scientific research, data analysis, and machine learning.

What tools can we use to program?

- Google Colab
- IDLE
- Terminal scripting
- VS Code

Demo

- Installing Python: https://python.org
- Using IDLE
- Terminal scripting and file systems
- Jupyter Notebooks and Google Colab

Some shell commands

- pwd: Print Working Directory
- ls: List files and folders
- Mkdir: Make a directory/folder -> mkdir foldername
- cd: change directory -> cd foldername
- cp: Copy files -> cp ogfilename newfilename
- mv: Move files -> mv filename1 foldername
- * : wildcard operator -> *.py refers to all Python files

Visual Studio Code (Optional)

- Download: https://code.visualstudio.com/download
- Click on the “Extensions” button on the left sidebar
- Search for “Python” and click “Install”

Jupyter Notebooks and Google Colab

- Google Colab: https://colab.google/
- Click on “New Notebook”
- Click on the box that says “Start coding…”, and print a message
- Eg. print(“Hello from Colab”)
- Click on the “Run” button, or press Shift+Enter

Follow along with Colab

Variables and data types

Math Operators

Boolean Logic

Comparison Operators

Conditionals

Slicing

Loops

What are some common data types?

Integers: Whole numbers

Eg. 5, -10

Floats: Numbers with decimals

Eg. 3.14, -0.5

Strings: Text enclosed in quotes

Eg. “Hello, World!”

Lists: A collection of items

Eg. [1, 2, 3] or ["A", "T", "G", "C"]

Booleans

Eg. True or False

Type casting

Type casting (or type conversion) is the process of converting a variable from one data type to another.

Function	Description	Example	Result
int()	Converts a value to an integer	int(3.14)	3
float()	Converts a value to a floating-point number	float(3)	3.0
str()	Converts a value to a string	str(123)	“123”
list()	Converts a value to a list (if possible)	list("hello")	['h', 'e', 'l', 'l', 'o']
bool()	Converts a value to a Boolean	bool(0) bool(“0”)	False True

What are variables?

A variable is like a container that stores information for your program to use later.

You can name the variable anything you want (with some syntax limitations)

How to create a variable in Python:

x = 10

name = “DNA Sequence”

The **variable** x now stores the **value** 10

The **variable** name now stores the **value** “DNA Sequence”

Try it in Colab!

- In a new cell:
- Create a variable age and store how old you are

  Hint: What data type is best to store an age?

- Create a variable hometown and store where you're from

  Hint: What data type is best to store a hometown?

- Print a sentence about how old you are and where you're from

print("I'm", age, "years old and from", hometown)

Or

print(f"I'm {age} years old and from {hometown}")

Recall:

x = 10

name = "DNA Sequence"

What are comparison operators?

Recall:

- A Boolean is True or False

Definition

- Comparison operators compare two values and return a Boolean value.

Note

- The assignment operator “=“ is not a comparison operator!

Operator	Meaning	Example
==	Equal to	5 == 5
!=	Not equal to	5 != 3
<	Less than	3 < 5
>	Greater than	10 > 7
<=	Less than or equal to	5 <= 5
>=	Greater than or equal to	7 >= 10

E.g Is the left hand value [some operator] the right hand value?

Try it in Colab!

Recall:

print(x < y)

Will print “True” or “False”

- In a new cell:
- Create variables x, y, and z and store three different numbers
- Print out the following results of each comparison:
  - x is greater than y
  - z is equal to y
  - y is not equal to x
  - z is less than and equal to y

Math Operators

Operator	Name	Example	Result
+	Addition	5 + 3	8
-	Subtraction	5 - 3	2
*	Multiplication	5 * 3	15
/	Division	5 / 3	1.6666666666666667
//	Floor Division	5 // 3	1
%	Modulus (Remainder)	5 % 3	2
**	Exponentiation	5 ** 3	125

```
x = 5
y = 3
z = x+y # z is now 8
x += y # x is now 8
```

x += y means the same thing as x = x+y

Try it in Colab!

Hint:
Convert the current time and lunch time to minutes and find the difference.
OR
Extract the hours and minutes from both times and find the difference.

- In a new cell:
- Create a variable now and store the current time in military format (e.g., 10:30 AM = 1030).
- Create another variable lunch and store the time for lunch (e.g., 1200 for noon)
- Calculate how many hours and minutes are left until lunch.
- print a message like: "1 hour and 30 minutes until lunch!"

Boolean values and Logical operators

Recall

- A Boolean value can only be either True or False

Definition

- Logical operators are used to combine or modify Boolean values, returning a Boolean result

operator	Description	Example:	Explanation
and	True if both are true	A and B	Both conditions must be true
or	True if at least one is true	A or B	At least one condition must be true
not	Reverses the Boolean value	not A	The opposite of the condition must be true

Boolean values and Logical operators

Operator	Description	Example: You're eligible for MSRP Bio …	Explanation
and	True if both are true	"If you're interested in research **and** have a minimum GPA of 3.5 in STEM courses"	Both conditions must be true
or	True if at least one is true	"If you're a full time undergraduate at a university **or** college"	At least one condition must be true
not	Reverses the Boolean value	"If you're **not** a freshman"	The opposite of the condition must be true

Try it in Colab!

Recall – Assigning values to variables:
interest_research = True

- Create the following variables with True /False values (except GPA, which should be a float)
  - interest_research
  - GPA
  - in_university
  - in_college
  - freshman
- Create a variable result, and assign it to the following:

  result = (interest_research and GPA >= 3.5) and (in_university or in_college) and (not freshman)
- Print the result!

  print(f"I should check out MSRP-Bio: {result}")

Conditionals

Definition

- Conditionals allow a program to execute different sections of code depending on the conditions.

Note

- if, elif, else
- Indentation is key!

```
if condition1:
    # Code to run if condition1 is true
```

Code on the first “level"

Code that is indented by one “level”
Indentation is done with the “Tab” key

Conditionals

Definition

- Conditionals allow a program to execute different sections of code depending on the conditions.

Note

- if, elif, else
- Indentation is key!

```
if condition1:
    # Code to run if condition1 is true
if condition2:
    # Code to run if condition2 is True
    # more code to run if condition 2 is True
```

- Code on the same “level"
- Code that is indented by one “level”
  Indentation is done with the “Tab” key

Conditionals

Definition

- Conditionals allow a program to execute different sections of code depending on the conditions.

Note

- if, elif, else
- Indentation is key!

```
if condition1:
    # Code to run if condition1 is true
elif condition2:
    # Code to run if condition2 is True and condition1 is False
    # more code to run if condition 2 is True
```

- Code on the same “level"
- Code that is indented by one “level”
  Indentation is done with the “Tab” key

Conditionals

Definition

- Conditionals allow a program to execute different sections of code depending on the conditions.

Note

- if, elif, else
- Indentation is key!

```
if condition1:
    # Code to run if condition1 is true
elif condition2:
    # Code to run if condition2 is True and condition1 is false
    # more code to run if condition 2 is True
else:
    #code to run only if all of the above conditions are false
```

- Code on the same “level"
- Code that is indented by one “level”
  Indentation is done with the “Tab” key

Conditionals

Definition

- Conditionals allow a program to execute different sections of code depending on the conditions.

Note

- if, elif, else
- Indentation is key!

```
if condition1:
    # Code to run if condition1 is true
elif condition2:
    # Code to run if condition2 is True and condition1 is false
    # more code to run if condition 2 is True
else:
    #code to run only if all of the above conditions are false

# A line of code written here will run after the if/elif/else block
```

Code on the same “level"

Code that is indented by one “level”
Indentation is done with the “Tab” key

Try it in Colab!

Remember this?

```
result = (interest_research and GPA >= 3.5) and (in_university or in_college) and (not freshman)
```

- In a new cell:
- Reuse your variables from earlier, and use conditional statements to print out something about MSRP-Bio!
- An idea:

```
if interest_research and GPA >= 3.5:

else:
    print ("There's still time to lock in!")
```

Try it in Colab!

Remember this?

```
result = (interest_research and GPA >= 3.5) and (in_university or in_college) and (not freshman)
```

- In a new cell:
- Reuse your variables from earlier, and use conditional statements to print out something about MSRP-Bio!
- An idea:

```
if interest_research and GPA >= 3.5:
    if in_university or in_college:

    else:
        print ("MSRP is for undergrads!")
else:
    print ("There's still time to lock in!")
```

Try it in Colab!

Remember this?

```
result = (interest_research and GPA >= 3.5) and (in_university or in_college) and (not freshman)
```

- In a new cell:
- Reuse your variables from earlier, and use conditional statements to print out something about MSRP-Bio!
- An idea:

```
if interest_research and GPA >= 3.5:
    if in_university or in_college:
        if not freshman:

        else:
            print ("I won't be a freshman forever!")
    else:
        print ("MSRP is for undergrads!")
else:
    print ("There's still time to lock in!")
```

Try it in Colab!

Remember this?

```
result = (interest_research and GPA >= 3.5) and (in_university or in_college) and (not freshman)
```

- In a new cell:
- Reuse your variables from earlier, and use conditional statements to print out something about MSRP-Bio!
- An idea:

```
if interest_research and GPA >= 3.5:
    if in_university or in_college:
        if not freshman:
            print ("I should check out MSRP Bio!")
        else:
            print ("I won't be a freshman forever!")
    else:
        print ("MSRP is for undergrads!")
else:
    print ("There's still time to lock in!")
```

What is indexing?

Recall:

- Strings are strings of characters, like “ATGCGTACG”
- Lists are collections of items, like [1, 2, 3] or ["A", "T", "G", "C"]

Definition

- Indexing is one way to extract a specific element of a sequence, like a string or a list

Note

- Indexing begins from 0 (first item on the left), or -1 (first item on the right)

What is indexing?

Recall:

- Strings are strings of characters, like “ATGCGTACG”
- Lists are collections of items, like [1, 2, 3] or ["A", "T", "G", "C"]

Definition

- Indexing is one way to extract a specific element of a sequence, like a string or a list

Note

- Indexing begins from 0 (first item on the left), or -1 (first item on the right)

dna = “ATGCGTACG”

What is indexing?

Recall:

- Strings are strings of characters, like “ATGCGTACG”
- Lists are collections of items, like [1, 2, 3] or ["A", "T", "G", "C"]

Definition

- Indexing is one way to extract a specific element of a sequence, like a string or a list

Note

- Indexing begins from 0 (first item on the left), or -1 (first item on the right)

dna = “ATGCGTACG”

Index: 0 1 2 3 4 5 6 7 8

What is indexing?

Recall:

- Strings are strings of characters, like “ATGCGTACG”
- Lists are collections of items, like [1, 2, 3] or ["A", "T", "G", "C"]

Definition

- Indexing is one way to extract a specific element of a sequence, like a string or a list

Note

- Indexing begins from 0 (first item on the left), or -1 (first item on the right)

dna = “ATGCGTACG”

Index: 0 1 2 3 4 5 6 7 8

… -3 -2 -1

What is indexing?

Recall:

- Strings are strings of characters, like “ATGCGTACG”
- Lists are collections of items, like [1, 2, 3] or ["A", "T", "G", "C"]

Definition

- Indexing is one way to extract a specific element of a sequence, like a string or a list

Note

- Indexing begins from 0 (first item on the left), or -1 (first item on the right)

dna = “ATGCGTACG”

Index: 0 1 2 3 4 5 6 7 8

… -3 -2 -1

dna[0] == “A”

dna[3] == “C”

dna[-1] == “G”

dna[-4] == “T”

What is slicing?

Recall:

- Strings are strings of characters, like “ATGCGTACG”
- Lists are collections of items, like [1, 2, 3] or ["A", "T", "G", "C"]

Definition

- Slicing is one way to extract items from a sequence, like a string or a list

Note

- Slicing uses the format start, stop, step

dna = “ATGCGTACG”

Index: 0 1 2 3 4 5 6 7 8

What is slicing?

Recall:

- Strings are strings of characters, like “ATGCGTACG”
- Lists are collections of items, like [1, 2, 3] or ["A", "T", "G", "C"]

Definition

- Slicing is one way to extract items from a sequence, like a string or a list

Note

- Slicing uses the format start, stop, step

dna = “ATGCGTACG”

Index: 0 1 2 3 4 5 6 7 8

dna[2:6] == “GCGT”

start:stop

What is slicing?

Recall:

- Strings are strings of characters, like “ATGCGTACG”
- Lists are collections of items, like [1, 2, 3] or ["A", "T", "G", "C"]

Definition

- Slicing is one way to extract items from a sequence, like a string or a list

Note

- Slicing uses the format start, stop, step

numbers = [1,2,3,4,5]

Index: 0 1 2 3 4

What is slicing?

Recall:

- Strings are strings of characters, like “ATGCGTACG”
- Lists are collections of items, like [1, 2, 3] or ["A", "T", "G", "C"]

Definition

- Slicing is one way to extract items from a sequence, like a string or a list

Note

- Slicing uses the format start, stop, step

numbers = [1,2,3,4,5]

Index: 0 1 2 3 4

numbers[:4] == [1,2,3,4]

What is slicing?

Recall:

- Strings are strings of characters, like “ATGCGTACG”
- Lists are collections of items, like [1, 2, 3] or ["A", "T", "G", "C"]

Definition

- Slicing is one way to extract items from a sequence, like a string or a list

Note

- Slicing uses the format start, stop, step

numbers = [1,2,3,4,5]

Index: 0 1 2 3 4

numbers[::2] == [1,3,5]

When left blank, here start and stop are left as their “defaults”, so the entire list is considered
The step size defaults to 1 if left blank

Try it in Colab!

- In a new cell:
- Create a list variable called numbers that holds 10 unique numbers
- Print out the same list, but use slicing so that only every third number is displayed.

For example, if the numbers list looks like: [1,2,3,4,5,6,7,8,9,10]
Your code should display: [3,6,9]

Hint: remember that indexing starts from 0, and the format of slicing is [start:stop:step] where start is **inclusive**, but stop is **exclusive**.

For loops

Definition

- For loops allow you to execute a block of code repeatedly for a fixed number of times

```
for i in range(5):
    print(i)
```

Prints out:
0
1
2
3
4

Start,stop,step

```
for i in range(1,11,3):
    print(i)
```

Prints out:
1
4
7
10

For loops

Definition

- For loops also allow you to “visit” every item in a collection

```
for item in [1,2,3,4]:
    print(item)
```

Prints out:
1
2
3
4

```
for letter in “Python”:
    print(letter)
```

Prints out:
P
y
t
h
o
n

Try it in Colab!

Remember:
x +=y means the same thing as x = x+y

- In new cells:
- Write a for loop that prints out every multiple of 5 between 5 and 100 (inclusive)
- Write a for loop that prints out every odd indexed number in the numbers list
- Write a for loop that prints out every even number in the numbers list

Hints:
You can find the length of a list with the **len()** function.
Example:

```
numbers = [1,2,3]
print(len(numbers)) #prints 3
```

While Loops

Definition

- While loops allow you to execute a block of code repeatedly until a condition is met

```
while condition1:
    # code to be executed
# Code to be executed when condition1 is false
```

```
x = 1
while x <= 5:
    print(f"Count: {x}")
    x += 1
```

```
x = 0
while True:
    print(x)
    x += 1
    if x == 10:
        print("Stopping the loop.")
        break
```

Try it in Colab!

Remember:
x +=y means the same thing as x = x+y

- In a new cell:
- Create a variable list called numbers that holds 10 unique numbers
- Create a variable called total and store the value 0
- Use a for loop to find the sum of all the numbers
- Use a while loop to find the sum of all the numbers

Hints:
You can find the length of a list with the len() function.
Example:

```
numbers = [1,2,3]
print(len(numbers))
```

Questions?

Variables and data types

Math Operators

Boolean Logic

Comparison Operators

Conditionals

Slicing

Loops

Up Next

Kahoot – 20m

(Time permitting) AMA fr


Kahoot!!!

After Lunch (till 1:45pm)

Dictionaries

Functions

Reading and writing to files

Pandas

Kahoot

Programming time !

Welcome Back!

Dictionaries

Definition

- A dictionary is a collection of key-value pairs. Each key is unique and is used to access its corresponding value.

dictionary = {key1:value1, key2:value2}

dictionary[key1] == value1

Dictionaries

Definition

- A dictionary is a collection of key-value pairs. Each key is unique and is used to access its corresponding value.

dictionary = {key1:value1, key2:value2}

inventory = {“Apple”:0.39, “Banana”:0.26}

inventory[“Banana”] == 0.26

Dictionaries

Definition

- A dictionary is a collection of key-value pairs. Each key is unique and is used to access its corresponding value.

dictionary = {key1:value1, key2:value2}

inventory = {“Apple”:0.39, “Banana”:0.26}

scores = {“Alice”:[88.5,92.3,85.0], “Bob”:[75.5, 80.0, 78.8]}

amino_acids = {"ATG": "Methionine", ”GCC": ”Alanine”}

Dictionaries

Definition

- A dictionary is a collection of key-value pairs. Each key is unique and is used to access its corresponding value.

```
dictionary = {key1:value1, key2:value2}

inventory = {“Apple”:0.39, “Banana”:0.26}

print(inventory.keys())
print(inventory.values())
print(inventory.items())
```

Try it in Colab!

Hint:
If the key doesn't exist in the dictionary yet, we can add a new key-value pair with:
dictionaryname[newkey] = newvalue

- In a new cell:
- Create a dictionary variable called codons that holds
- {"ATG": "Methionine", "CGG": "Arginine"}
- Print out the value associated with the key "ATG"
- Add a new key-value pair: "GCC": "Alanine"
- Loop through the keys and values using .items() to print out:
- codon : amino_acid

```
for key, value in dictionaryname.items()
    # do something with the key and value
```

Functions

Definition

- A function is a block of reusable code that performs a specific task. E.g. print()

```
def function_name():
    # code to do something
```

Functions

Definition

- A function is a block of reusable code that performs a specific task. E.g. print()

```
def function_name():
    # code to do something

function_name() # “calls” the function
```

Functions

Definition

- A function is a block of reusable code that performs a specific task. E.g. print()

```
def function_name(parameter):
    # code to do something with the parameter
```

Functions

Definition

- A function is a block of reusable code that performs a specific task. E.g. print()

```
def function_name(parameter):
    # code to do something with the parameter

function_name(some_input) # “calls” the function
```

Functions

Definition

- A function is a block of reusable code that performs a specific task. E.g. print()

```
def countdown(number):
    for i in range(number,-1,-1):
        print(i)

countdown(10)    # “calls” the function with the argument 10
```

Functions

Definition

- A function is a block of reusable code that performs a specific task. E.g. print()

```
def countdown(number):
    for i in range(number,-1,-1):
        print(i)

countdown(10)
```

Defines a function countdown that takes a parameter "number"

"calls" the function with the argument 10

Functions

Definition

- A function is a block of reusable code that performs a specific task. E.g. print()

```
                    10
def countdown(number):
    for i in range(number,-1,-1):
        print(i)

countdown(10)
```

Functions

Definition

- A function is a block of reusable code that performs a specific task. E.g. print()

```
                    10
def countdown(number):
    for i in range(number,-1,-1):
                                10
        print(i)

countdown(10)
```

Functions

Definition

- A function is a block of reusable code that performs a specific task. E.g. print()

```
                    10
def countdown(number):
    for i in range(number,-1,-1):
                                  10
        print(i)

countdown(10)
```

Output:

```
10
9
8
7
6
5
4
3
2
1
0
```

Functions

Definition

- A function is a block of reusable code that performs a specific task. E.g. print()

```
def double(number):
    return number * 2
```

Functions

Definition

- A function is a block of reusable code that performs a specific task. E.g. print()

```
def double(number):
    return number * 2

result = double(5)   # calls the function with the argument 5 and saves it to variable result

print(result)
```

Functions

Definition

- A function is a block of reusable code that performs a specific task. E.g. print()

```
                    5
def double(number):
    return number * 2

result = double(5)   # calls the function with the number 5
                     # and saves it to variable result

print(result)
```

Functions

Definition

- A function is a block of reusable code that performs a specific task. E.g. print()

```
                    5
def double(number):
    return number * 2    5 * 2 = 10

result = double(5)    # calls the function with the number 5
                      # and saves it to variable result

print(result)
```

Functions

Definition

- A function is a block of reusable code that performs a specific task. E.g. print()

```
                    5
def double(number):
    return number * 2    5 * 2 = 10

                10
result = double(5)   # calls the function with the number 5
                     # and saves it to variable result

print(result)
```

Functions

Definition

- A function is a block of reusable code that performs a specific task. E.g. print()

```
                    5
def double(number):
    return number * 2    5 * 2 = 10

                10
result = double(5)   # calls the function with the number 5
                     and saves it to variable result

          10
print(result)
```

Functions

Definition

- A function is a block of reusable code that performs a specific task. E.g. print()

```
                    5
def double(number):
    return number * 2    5 * 2 = 10

                 10
result = double(5)    # calls the function with the number 5
                      # and saves it to variable result
          10
print(result)
```

Output:
10

Try it in Colab!

Remember:
len(some_string) returns the length of a string

Hint:
some_string.count(“G”) returns the number of “G”s in the string

- In a new cell:
- Define a function calculate_gc_content that takes a string parameter “dna”
- In the function, count the number of “G”s and “C”s in dna and save them in two variables.
- Calculate the percentage of gc content against the length of the dna
- Return the result
- Call the function with "ATGCGTAC" and print out the result

$$\frac{(g\_count + c\_count)}{\text{length of the dna}} \times 100$$

File I/O in Python

Definition

- File I/O refers to reading data from or writing data to files

```
# Reading from a file
with open("example.txt", "r") as file:
    content = file.read()
    print(content)
```

r – Read mode

File I/O in Python

Definition

- File I/O refers to reading data from or writing data to files

```
# Writing to a file
with open("example.txt", "w") as file:
    file.write("This is a sample file.")
```

w – Write mode

File I/O in Python

Definition

- File I/O refers to reading data from or writing data to files

```
# Reading from a file
with open("example.txt", "r") as file:
    content = file.read()
    print(content)
```

r – Read mode

Try it in Colab!

- In a new cell:
- Write to a new file, sequences.txt, the following 3 lines:
  ATG
  ATGCGTACGTT
  GCTAGCTAGCT
- Open the input file sequences.txt in read mode
  - Open an output file filtered.txt in write mode
    - Loop through each line in the input file
      - If the sequence length is greater than 10, write it to filtered.txt

Hint:

```
with open("example.txt", "r") as file:
    content = file.read() #reads all the content
    for line in file:
        #reads the file line by line
with open("example.txt", "w") as file:
    file.write("text")
```

Pandas (and csvs)

Definition

- Pandas is a Python library for data manipulation and analysis.
- A common file type you'll be working with is .csv

A	B	C	D	E
Name	Math	Science	English	History
Alice	88	91	87	85
Bob	75	78	80	77
Charlie	92	85	89	88
Diana	85	90	85	92
Ethan	79	84	90	86

A CSV (Comma separated value) opened with Excel

```
Name,Math,Science,English,History
Alice,88,91,87,85
Bob,75,78,80,77
Charlie,92,85,89,88
Diana,85,90,85,92
Ethan,79,84,90,86
```

The truth

Pandas (and csvs)

Definition

- Pandas is a Python library for data manipulation and analysis.
- A common file type you'll be working with is .csv

```
import pandas as pd
```

Pandas (and csvs)

Definition

- Pandas is a Python library for data manipulation and analysis.
- A common file type you'll be working with is .csv

```
import pandas as pd
```

e.g. "grades.csv"

```
dataframe = pd.read_csv(filename)
```

A 2D labeled data structure

Pandas (and csvs)

Definition

- Pandas is a Python library for data manipulation and analysis.
- A common file type you'll be working with is .csv

```
import pandas as pd

dataframe = pandas.read_csv(filename)
```

Pandas (and csvs)

Definition

- Pandas is a Python library for data manipulation and analysis.
- A common file type you'll be working with is .csv

```
import pandas as pd
```

e.g. "grades.csv"

```
dataframe = pd.read_csv(filename)
```

Pandas (and csvs)

```
import pandas as pd

df = pd.read_csv("grades.csv")

df.head() # the first 5 rows
df.head(10) #the first 10 rows
```

```
Name,Math,Science,English,History
Alice,88,91,87,85
Bob,75,78,80,77
Charlie,92,85,89,88
Diana,85,90,85,92
Ethan,79,84,90,86
```

grades.csv

Pandas (and csvs)

```
import pandas as pd

df = pd.read_csv(“grades.csv”)

df[“Science”] # the Science column only
df[[“Science”,”Math]] # the Science and Math column
```

```
Name,Math,Science,English,History
Alice,88,91,87,85
Bob,75,78,80,77
Charlie,92,85,89,88
Diana,85,90,85,92
Ethan,79,84,90,86
```

grades.csv

Pandas (and csvs)

```
import pandas as pd

df = pd.read_csv(“grades.csv”)

df[“Sum”] = df[“Math”] + df[”Science”] # makes a new column
```

```
Name,Math,Science,English,History
Alice,88,91,87,85
Bob,75,78,80,77
Charlie,92,85,89,88
Diana,85,90,85,92
Ethan,79,84,90,86
```

grades.csv

Pandas (and csvs)

```
import pandas as pd

df = pd.read_csv(“grades.csv”)

df = df.drop(columns=[”Math”, Science])
df = df.drop(“Math”)
```

#drops both columns

```
Name,Math,Science,English,History
Alice,88,91,87,85
Bob,75,78,80,77
Charlie,92,85,89,88
Diana,85,90,85,92
Ethan,79,84,90,86
```

grades.csv

Pandas (and csvs)

```
import pandas as pd

df = pd.read_csv(“grades.csv”)

df[”English”].mean() # the average English grades
df[”English”].max() # the highest English grade
df[”English”].min() # the lowest English grade
df[”English”].std() # The standard deviation for English
```

```
Name,Math,Science,English,History
Alice,88,91,87,85
Bob,75,78,80,77
Charlie,92,85,89,88
Diana,85,90,85,92
Ethan,79,84,90,86
```

grades.csv

Try it in Colab!

- In a new cell:
- import pandas as pd
- Create a dataframe df by reading california_housing_test.csv (in colab's sample_data folder)
- Print the first 10 rows
- Drop the longitude and latitude columns
- Print the first 10 rows again
- Calculate the average number of rooms per household and make a new column avg_rooms

Hint:
df = df.drop(columns = [col1,col2…])

Try it in Colab! (cont)

Hint:
We can filter data with logical/comparison operators:
filtered = df[df[“median_house_value”] > 150000]

- Filter rows where median house value is greater than $150,000 and save that as a new dataframe variable “filtered”
- Print the first few rows of the filtered dataframe
- save the filtered dataframe to the file “filtered_housing_data.csv”

Hint:
Convert pandas dataframes to csvs with:
filtered.to_csv(“filtered_housing_data.csv”)

Questions?

Dictionaries

Functions

Reading and writing to files

Pandas

KAHOOT Part 2!!

Programming time!
(Until 4:30pm)

Thank you!

Keep in touch:

gwoo@mit.edu

I am also on LinkedIn (Georgina Woo)