



CS 1112: Introduction To Programming

Match-Case statement;

Mutability in Python

Dr. Nada Basit // `basit[at]Virginia[dot]edu`

Friendly Reminders

- Your **safety** and **comfort** is important!
 - If you choose to wear a mask you are welcome to do so
 - *We will interpret wearing a mask as being considerate and caring of others in the classroom (not that you are sick), and realize that some may choose to mask to remain distanced*
- Remember to always be **kind, respectful, supportive, compassionate** and **mindful of others!** 😊
- Be an **active** participant in your learning!
You're welcome and **encouraged** to ask questions during class!
- If you feel **unwell**, or think you are, **please stay home**
 - *Contact us! We will work with you!*
 - Get some rest 😊
 - View the recorded lectures – *please allow 24-48 hours to post*



Announcements

- **PA02** is out and is **due by 11:00pm on Wednesday (2/12)!**
 - Submit on Gradescope: your .py file and a reflection file (PDF).
 - *Not sure how to create/submit a **PDF**? No problem! Ask one of the TAs for help!*
- **Quiz 2** has been graded. Scores can be seen on **Sherlock** (will eventually be on Canvas).
- **Quiz 3** will be released this afternoon and is **due by 11:00pm on Monday (2/10)!**
 - No late quizzes accepted
 - No make-up quizzes allowed
 - If you believe your computer is glitching, it's a good idea to copy down your answers to each of the questions in a word document. In the event something happens, you can send me your solutions.
 - ***Note:** in general, will **cannot and will not** accept quiz solutions via **email**. We will only accept them in the case where your quiz may have glitched and we *no longer have your submitted answers*.*
 - **Take quiz on:** [Sherlock.cs.virginia.edu](https://sherlock.cs.virginia.edu)

Review Solution

`conditionals.ica.py`

See how we can write a `gpa_calc` function (*challenge portion of the activity*)

Match-Case statement

Alternative to **if-elif-else** statement

Match-Case statement

- In short, we pass a **parameter** then try to check in which **case** the parameter is satisfied
 - If we find a **match**, we will do something, and if there is no match at all we will do something else
- Initialized with the **match** keyword
 - Creating a block and taking a **parameter**
(here the variable name is also parameter)
- Cases are established with the **case** keyword
 - There are several cases followed by a **pattern**
 - In each case the **pattern** tries to match the **parameter**
- The “**_**” is the **wildcard character** which is run when nothing is matched

```
parameter = "SomeValue"

match parameter:
    case "first":
        # do something (first)
        print("first")
    case "second":
        # do something (second)
        print("second")
    case "third":
        # do something (third)
        print("third")
        # do more stuff
        # ...
    case "n":
        # do something (n)
        print("n")
    case _:
        # nothing else matched, do this now
        print("Nothing else matched!")
```

Examples

Note: “ | ” means “**or**”

```
character = 'M'
```

```
match character:
```

```
    case 'A' | 'Z':
```

```
        print("character is A or Z")
```

```
    case 'B' | 'D':
```

```
        print("character is B or D")
```

```
    case 'C' | 'M':
```

```
        print("character is C or M")
```

```
character = 'V'
```

```
match character:
```

```
    case 'A' | 'Z':
```

```
        print("character is A or Z")
```

```
    case 'B' | 'D':
```

```
        print("character is B or D")
```

```
    case 'C' | 'M':
```

```
        print("character is C or M")
```

```
    case _:
```

```
        print("Unknown character given")
```

More Examples (complex ones)

```
def alarm(item):  
    match item:  
        case ['evening', action]:  
            print(f'You almost finished the day! Now {action}!')  
        case [time, action]:  
            print(f'Good {time}! It is time to {action}!')  
        case _:  
            print('The time is invalid.')
```

```
alarm(['evening', 'play video games'])
```

```
You almost finished the day! Now play video games!
```

```
alarm(['evening', 'work'])
```

```
You almost finished the day! Now work!
```

```
def alarm(item):  
    match item:  
        case ['evening', action] if action not in ['work', 'study']:  
            print(f'You almost finished the day! Now {action}!')  
        case ['evening', _]:  
            print('Come on, you deserve some rest!')  
        case [time, action]:  
            print(f'Good {time}! It is time to {action}!')  
        case _:  
            print('The time is invalid.')
```

```
alarm(['evening', 'study'])
```

```
Come on, you deserve some rest!
```


More on Match-Case statements

- Snippet taken from [geeksforgeeks](#)



Q: How does the match-case statement differ from if-elif-else statements?

A: The match-case statement is a more powerful and expressive construct compared to if-elif-else statements. While if-elif-else statements rely on boolean expressions, match-case statements can match patterns based on the structure and value of the data. Match-case statements provide a more structured and readable way to handle multiple conditions and perform different actions based on those conditions.

Q: What are the benefits of using the match-case statement?

A: The match-case statement offers several benefits, including:

- **Conciseness:** Match-case statements allow you to express complex branching logic in a concise and readable manner.
- **Readability:** Pattern matching makes the code more readable and self-explanatory, as it closely resembles the problem domain.
- **Safety:** Match-case statements provide exhaustive pattern matching, ensuring that all possible cases are handled.



PYTHON DEMONSTRATION

Let's jump on PyCharm!

`match_case.py`

Mutability

Mutability

- In Python, as in some other languages, data structures are either *mutable* or *immutable*.
 - **Mutable**: the values of the structure *can* be changed
 - **Immutable**: the values of the structure *cannot* be changed (the opposite)
- Learning which structures are which is **vital for writing good code** and selecting appropriate data structures to use. Additionally, you can be mindful of how to use immutable data types.
- A **list** is a *data structure* that is **mutable**
- The **string** *data type* is **immutable**

Mutability (think “mutate” ... changeable)

- A collection is **mutable** if I can **reassign** a value at an index **without** creating a **new copy** of the collection.
- A collection is **immutable** if it **can not** be **changed** without creating a brand **new copy** of the collection.
- A **list** is a *data structure* that is **mutable**
- The **string** *data type* is **immutable**

Properties of some collections

<u>Type</u>	<u>Stores</u>	<u>Syntax</u>	<u>Mutable?</u>
Range	ints	<code>range(3, 7)</code>	no
String	characters	<code>"Hello", "abc 123"</code>	no
List	anything	<code>[1, 2, 3, 6, "hello"]</code>	yes
Tuple	anything	<code>(1, 2, 3, 6, "hello")</code>	no
Dictionary	key:value pairs	<code>{17: "hi", 29: "bye"}</code>	yes
Set	anything	<code>{1, 2, 6, "hi"}</code>	yes


Mutability as it relates to memory

- We can change a value stored in a *variable* that is of an **immutable** type
 - However, that new “value” is stored in a **new memory location**.
- We can get a variables memory location using `hex(id(var_name))`
- Example:

```
my_str = "Hello World"
print(my_str, "\ttype:", type(my_str), "\tstored at:", hex(id(my_str)))
my_str = my_str.upper()
print(my_str, "\ttype:", type(my_str), "\tstored at:", hex(id(my_str)))
```

- Prints: (notice the *address has changed*)

```
Hello World    type: <class 'str'>    stored at: 0x1eb39bf2530
HELLO WORLD    type: <class 'str'>    stored at: 0x1eb39bf5330
```



Mutability

```
my_str = "Hello World"
print(my_str, "\ttype:", type(my_str), "\tstored at:", hex(id(my_str)))
my_str = my_str.upper()
print(my_str, "\ttype:", type(my_str), "\tstored at:", hex(id(my_str)))
```

- We said...
- We can change a value stored in a *variable* that is of an **immutable** type
 - However, that new “value” is stored in a **new memory location** (*address*)

simulated
address

100	
200	“Hello World”
300	
400	

← my_str

simulated
address

100	
200	“Hello World”
...	
900	“HELLO WORLD”

← my_str

Mutability as it relates to memory

The same is **not** true for **lists**:

*If we change the first element of **my_list** to a 9...*

```
my_list = [8, 6, 7, 5, 3, 0, 9]
print(my_list, "\ttype:", type(my_list), "\tstored at:", hex(id(my_list)))
my_list[0] = 9
print(my_list, "\ttype:", type(my_list), "\tstored at:", hex(id(my_list)))
```

Prints:

(the address is **still the same!**)

```
[8, 6, 7, 5, 3, 0, 9]  type: <class 'list'>    stored at: 0x1eb39bf2c88
[9, 6, 7, 5, 3, 0, 9]  type: <class 'list'>    stored at: 0x1eb39bf2c88
```



Let's look at the **string** data type

- In Python, **strings are immutable**
- That is, they *cannot* be mutated or changed
- You can swap out a whole string for another (this is OK). You can assign strings to variables, and reassign new strings to the same variable, but individual *characters* within a string *cannot* be reassigned.
- Example:

```
1 | greeting = "hi there!"  
2 | greeting[0] = "H"  
3 | print(greeting)
```

```
Error: Line 2  
TypeError: 'str' does not support item assignment on line 2
```



PYTHON DEMONSTRATION

Let's jump on PyCharm!

`mutability_examples.py`

`mutability_exercise.py`

Reminder: CS Laptop Loaner Program

- This course requires students to have a **laptop**
- I realize that not everybody might have one (nor necessarily need one for their desired major / path...)
- If you do not have a laptop for any reason... *not to worry!*
- The CS department's Systems staff has a notebook / laptop loaner program and will be able to loan you a notebook / laptop computer for the duration of the semester if you don't have one or if you cannot afford one.
 - Also available if your laptop is broken and under repair, we can arrange for you to receive a loaner laptop for a week or two until your own laptop is fixed

Interested? Link: https://www.cs.virginia.edu/wiki/doku.php?id=cs_laptop_loaner

I am happy to be your sponsor. Please let me know.