Welcome back to CS 2100!

Prof. Rasika Bhalerao

Poll: How can I swap the values of `x: int` and `y: int`?

a.

```
temp: int = x
x = y
y = temp
```

b.

```
temp: int = x
temp = y
y = x
```

c.

```
temp: int = y
x = y
x = temp
```

d.

```
temp: int = x
y = temp
x = y
```

String manipulation

Strings in Python can be represented using single quotes (`'cat'`) or double quotes (`"cat"`).

Easy way to put a quote in a string:

- `'This is a double quote: "'`
- `"This is a single quote: '"`

Or use an escape sequence (indicate that it shouldn't end the string): `"This is a double quote: \""`

String escape sequences

- `"\t"` tab character
- `"\n"` newline character
- `"\r"` carriage return
- `"\""` double quote character
- `"\'"` single quote character
- `"\\"` backslash character

String concatenation

- `"Joseph" + "Aoun"`
- `"Joseph" + " " + "Aoun"`
- We can multiply a string by an integer: `"!" * 10`

Poll: What does this print? `print("I am so excited" + "!" * 3)`

1. I am so excited! I am so excited! I am so excited!
2. I am so excited!I am so excited!I am so excited!
3. I am so excited!!!
4. Nothing – it breaks

F-string

We can put a variable directly in a string to save time.

```python
cats: int = 4
print(f"There are {cats} cats in this room.")
```

```
There are 4 cats in this room.
```

It also helps reduce the number of places where a bug can happen in the code.

Float

We've seen: `int`, `str`

What if we want to represent a number that isn't an integer? Like 2.5? Use a `float`.

```
num: float = 2.5
```

Tip: dividing any two numbers in Python results in a float -- even if the operands are `int`s, and the result has an integer value.

```
print(type(4 / 2))
```

```
<class 'float'>
```

Boolean

- Store the result of a yes-no question
- Only 2 (legit) values: True and False
- Assuming you've seen booleans before

```
my_decision: bool = True
```

- Opposite
  - `not my_decision`
- Comparisons: `<`, `<=`, `>`, `>=`, `==`, `!=`
  - `4 < 6`
- And (both must be true to result in true)
  - `my_decision and your_decision`
- Or (true if either/both are true)
  - `my_decision or your_decision`

Order of operations

- Math order of operations:

  2. (Parentheses)
  3. Exponents
     - `4**2 * 3`

       `>> 48`

  iv. Multiplication/Division (left to right)

  v. Addition/Subtraction (left to right)

- Math happens before comparison operations
  - `7 < 2 + 8`

    `>> True`
- Comparison happens before boolean operations
  - `3 < 4 and 5 < 7`

    `True`

None

`None` works like a value that represents the absence of a value.

```
bodyguard_name: str = None # doesn't have a value -- I don't have a bodyguard
```

It's different from "" or 0 (see Null Island)

Can store `None` in a list

```
grades: List[int] = [5, None, 0]
```

Cannot add `None` to a number or string

- `None + "hi"` does not work
- `len(None)` does not work

Use `Optional` to specify that a type might be `None`

```python
from typing import Optional

def get_number_or_None(hopefully_a_number: str) -> Optional[int]:
    try:
        return int(hopefully_a_number)
    except ValueError:
        return None
```

Lists and sets

```python
from typing import List, Set

nums: List[int] = [1, 2, 3]
words: Set[str] = {'hi', 'hello', 'howdy'}
```

Assuming you have used lists before. Sets may be new to some.

A set is very similar to a list: it is a collection of items.

Differences between set and list:

- A set is unordered
- A set can only hold each item (at most) once -- no duplicates

There will be more about sets in Lecture 12.

Control structures

We're assuming you've seen conditionals and iteration before, though possibly in a different programming langauge.

Here it is in Python.

Conditionals: if / else

```python
secret_num: int = 8
guess: int = int(input('My guess: '))

if secret_num == guess:
    print('I guessed it!')
elif (secret_num + 1 == guess) or (secret_num - 1 == guess):
    print('So close!')
else:
    print('Maybe next time!')
```

Tip: we can put a conditional expression in one line:

```python
print('yes' if my_decision else 'no')
print(f'{num_cats} cat{'s' if num_cats > 1 else ''}')
```

Conditionals: match case statements

Many cases --> match-case statement might be more practical

```python
name: str = input('Please enter your name: ')
match name:
    case 'SpongeBob':
        print('You are a sponge')
    case 'Patrick':
        print('You are a starfish')
    case _:
        print('I don\'t know you')
```

- Finds the first case that matches, and only executes that one case
    - or zero cases if none match
- `case _` is a catch-all that matches anything that didn't fit any other cases
    - Not required, but if it is there, it must be the last case

Iteration: while loops

```python
animal: str = input('Please enter an animal: ')

while not is_animal(animal):
    animal = input('That wasn\'t an animal. Please enter an animal: ')
```

Iteration: for loops over numbers

Don't know in advance how many iterations --> while loop

Know the number of iterations (given the variables we currently have) --> for loop

Helpful function: `range()`

```
for i in range(4):
    print(i)

>> 0
   1
   2
   3
```

Can start at a number other than 0:

```
for i in range(2, 5):
    print(i)

>> 2
   3
   4
```

Can count in "steps" larger than 1:

```
for i in range(10, 50, 5):
    print(i)

>> 10
   15
   20
   25
   30
   35
   40
   45
```

Iteration: for loops over the elements of a collection

The `range()` function returns a collection, which the for loop iterates over.

Can iterate over the elements of a different collection:

```
for character in 'I love cats!':
    print(character.upper())
```

```
I
 
L
O
V
E
 
C
A
T
S
!
```

```
def sarcasm(phrase: str) -> str:
    """Returns the sarcastic version
    of the provided phrase, where a
    randomly selected half of the
    characters are uppercase, and
    the others are lowercase.

    Parameters
    ----------
    phrase : str
        The phrase to turn sarcastic
    Returns
    -------
    str
        The sarcastic version of
        the phrase
    """
    sarcastic_phrase = ''
    for character in phrase:
      if random() < 0.5:
        sarcastic_phrase +=
            character.upper()
    return sarcastic_phrase
```

Poll: What's wrong? Why doesn't the docstring match the code?

1. It's adding the index of the character, not the character itself
2. It skips adding about half of the letters
3. Sometimes, it doesn't return a string at all
4. It adds extra characters to the string

Poll: Which of these is a one-line version of the inside of the (correct) `sarcasm()` function?

1. `return ''.join([character.upper() for character in phrase if random() < 0.5])`
2. `return ''.join([character.upper() if random() < 0.5 for character in phrase])`
3. `return ''.join([character.upper() if random() else character.lower() for character in phrase])`
4. `return ''.join([character.upper() if random() < 0.5 else character.lower() for character in phrase])`

Iteration: for loops over a collection, keeping track of indices

```
for index, word in enumerate(['American Shorthair', 'Balinese', 'Cheetah']):
    print(f'{index}: {word}')

>> 0: American Shorthair
   1: Balinese
   2: Cheetah
```

Read and write data from text files

Read data from a file:

```python
with open('story.txt', 'r', encoding="utf-8") as file:
    for line in file.readlines():
        print(line)
```

Write to a file instead of reading --> use an option other than `'r'`:

- `open('story.txt', 'r')`: read the file
- `open('story.txt', 'w')`: write the file (overwrite it if it already exists)
- `open('story.txt', 'a')`: append to the end of the file (and create the file if it doesn't exist)

Write to the file using `file.write("Line to write to file")`.

Import code

Import modules like `import unittest` and `from typing import List, Set`

Can also import code from a file that we wrote ourselves: `import my_file`

When a Python file is imported, all of the code inside it is executed. That's why we put our code inside functions -- we don't want the code inside to be executed when it's imported!

Call all the functions in `main()` and add this at the end of the file:

```
if __name__ == '__main__':
    main()
```

Import code: trying it out in lecture

1. Put `print('hello')` in a new file, import it in this file, and run this file (where it is imported)
2. Move it to inside a function and run it
3. Add the `if __name__ == '__main__'` conditional at the end and run it
4. Run this file (where it is imported) to make sure it doesn't print

Poll:

1. What is your main takeaway from today?
2. What would you like to revisit next time?