Quiz 1 Review Session

Content on Quiz 1

- Cumulative! New topics build on old ones
- Boolean Operators
- Conditions (if/else/elif)
- f-strings, Positional Arguments
- Recursion
- Named Constants, Default Parameters

Disclaimer: We haven’t seen the quiz; this review session covers the main topics in the unit.

Boolean Operators

- or - **either** is true
- and - **both** are true
- not - **negating**: True becomes False, False becomes True

Precedence (highest to lowest):

0. Arithmetic operators (PEMDAS)
1. Relational Operators
2. Not
3. And
4. Or

Boolean Operators

What is the result of the following expressions?

- ➔ not True and not False False
- ➔ 3 + 4 < 5 or 5 - 4 == 3 False
- ➔ True or False and not (False and True) True
- ➔ "A" == "a" or "B" == "B" True

Conditionals

- *if* and *elif* statements must be followed by a boolean condition
- The condition must be simplified
- The *then* block following *if, elif, and else* blocks are indented
- After completing the *then* block of one *if*, *elif*, or *else* block, you continue onto the next statement after the *if-else*

```
if <condition>:
    <then, execute these statements>
else:
    <execute these other statements>
<rest of program>
```

Conditionals

True or False?

- ➔ For every *if*, there needs to be an *else* False
- ➔ For every *else*, there needs to be an *if* True
- ➔ The condition following *if* and *elif* must be a numerical expression False, boolean
- ➔ *elif* statements help us simplify code, making it easier to read True

- ➔ How many *if's* can I have for an *else*? 1
- ➔ How many *elif's* can I have for an *if*? 0, or infinite
- ➔ How many *else's* can I have for an *if*? 1

f-strings

- Helps us format strings easily by allowing us to embed expressions directly into the string

```
print(f"Quiz {0 + 1} is in {14 - 12} days!")
```

Keyword Arguments

Previously:

```
def sum(num1: int, num2: int) -> int:
    sum(num1 = 11, num2 = 3)
```

Keyword arguments:

- assigning values based on parameter names
- order doesn't matter!

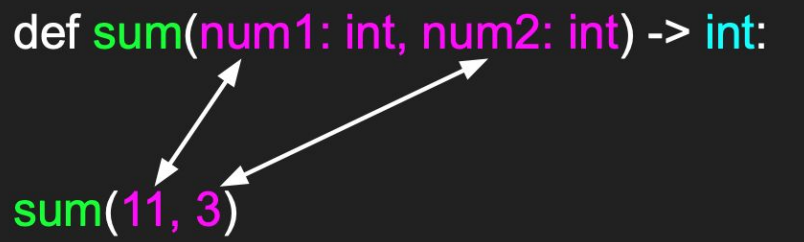
```
sum(num1 = 11, num2 = 3)
```

```
sum(num2 = 3, num1 = 11)
```

Positional Arguments

As opposed to positional arguments;

- Values are assigned based on the order (position) of arguments
- order **does** matter



How do you tell positional and keyword arguments apart?

- positional: if **arguments** in a function call only contains values
- keyword: if **parameter names** appear in the function call

Recursion

- Calling a function within itself, or multiple functions repeatedly call each other
- Made up of (at least) one base case and (at least) one recursive case
- Base case: a branch where the function stops, and does not recur
- A recursive case must make progress towards the base case
  - Progress is typically made by changing the argument of the recursive call so that the argument gets closer to the base case
- Infinite recursion results in a RecursionError or StackOverflowError

```
def safe_icarus(x: int) -> int:
    """Bound aspirations!"""
    if x >= 2:
        return 1
    else:
        return 1 + safe_icarus(x=x + 1)
```

Recursion

True or False?

- ➔ The base case allows a recursive function to stop — True
- ➔ A recursive case should make progress away from the base case — False

Is there anything wrong with the following code?

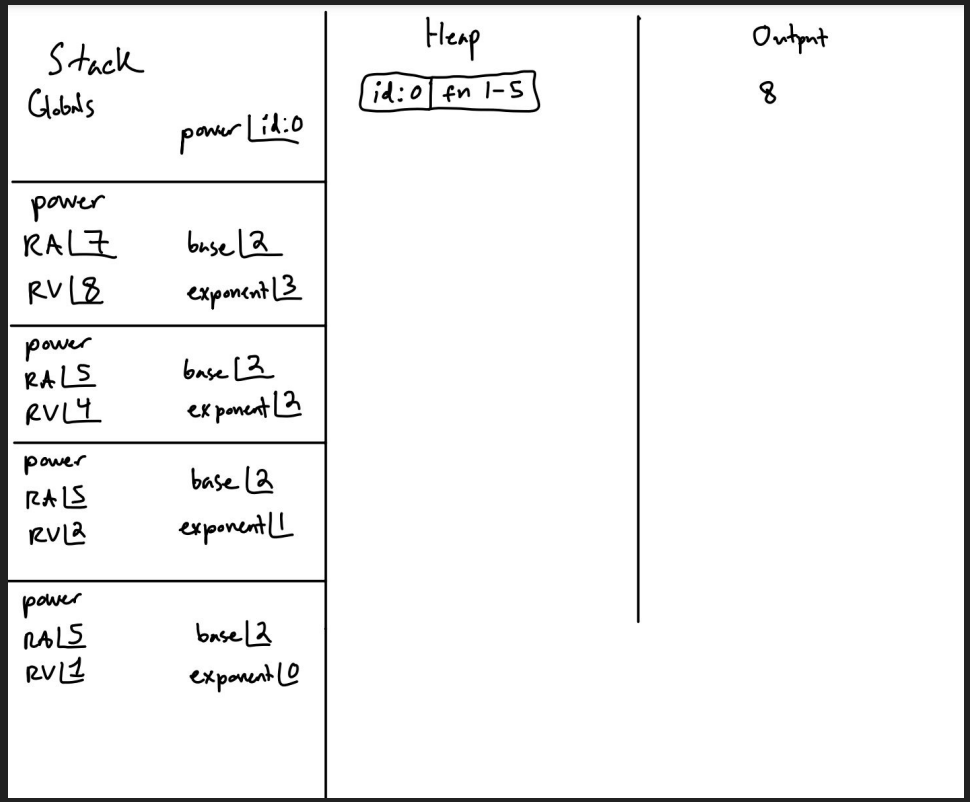
```
def factorial_ish(n: int) -> int:
    """Return the factorial of a number!"""
    if n == 0 or n == 1:
        return 1
    return (n * factorial_ish(n) - 1)
```

Yes - the recursive call `factorial_ish(n)` does **not** make progress towards the base case. Something like `factorial_ish(n)` does make progress towards the base case

```python
def power(base, exponent) -> int:
    """Return base raised to the power of exponent computed recursively."""
    if exponent == 0:
        return 1
    return base * power(base, exponent - 1)

print(power(2, 3))
```

Solution


Stack
Globals
power | id:0
power
RA | 7
RV | 8
base | 2
exponent | 3
power
RA | 5
RV | 4
base | 2
exponent | 2
power
RA | 5
RV | 2
base | 2
exponent | 1
power
RA | 5
RV | 1
base | 2
exponent | 0
Heap
id:0
fn 1-5
Output
8

Named Constants + Default Parameters

Named Constants:

- Hold the same value throughout the entire program
- Naming convention: ALL_CAPS, with underscores between words

Default Parameters:

- A **parameter** in a function signature that is set to a value
- If the function call does not include an argument value for that parameter, we use the default value
- Should always come after any non-default parameters

Default Parameters

Given the function signature, are the following function **calls** valid?

```
def study_or_not(days_left: int, am_lazy: bool, target_grade: int = 100) -> bool:
```

1. `study_or_not(True, 2, 100)` No

2. `study_or_not(am_lazy=False, days_left=2)` Yes

3. `study_or_not(2, True)` Yes

Code Writing Practice

Write a function called `study_or_not` that takes in three parameters and matches the following criteria:

- One `int` parameter called `days_left`, one `str` parameter called `am_lazy`, one `int` parameter called `target_grade`.
- If `am_lazy` is "Yes":
  - if the target grade is higher than 75 or you have less than 4 days left, return "Yes!"
  - if not, return "Take a break!"
- Otherwise, return "Yes!"

we strongly encourage studying

Write a call to the function so that "Take a break!" returns.

One Solution:

```
def study_or_not(days_left: int, am_lazy: str, target_grade: int) -> str:
    if am_lazy == "Yes":
        if target_grade > 75 or days_left < 4:
            return "Yes!"
        else:
            return "Take a break!"
    return "Yes!"

study_or_not(days_left=5, am_lazy="Yes", target_grade=70)
```

Questions?

Other Resources!

- Practice quiz on the course site with answers and explanations
  - We would recommend trying the problems out on your own, then checking your answers
- Tutoring
  - Thursday 3 - 5 in FB 141
- Office Hours
  - Tomorrow and Friday 11 - 5 in SN008