DISCUSSION 01

Control, Environment Diagrams

Mingxiao Wei
mingxiaowei@berkeley.edu

Sep 1, 2022

LOGISTICS 🏡

- Homework 01 due today 09/01 @ 11:59pm
- Hog is released! 🎲🎲🎲
  - You can choose to work alone or in group of 2
  - Checkpoint 1 due next Tue 09/06
  - The whole project due next Fri 09/09
  - Submit the whole project by next Thu 09/08 for one extra point!
- About office hours:
  - My OH is listed on our section website
  - Before the first midterm, my OH varies weekly AND both online and in-person ones are queue-based - just drop in and make a ticket, no appointment needed
  - Check out the OH calendar for the most updated OH schedule!

LOGISTICS 🏡

- One-time tutorial section signups are open! (Check out Ed post #176 for more info)
    - Tutorials are smaller sections (~5) where the tutor/mentor will go over some additional practice problems
    - Signups for recurring sections will open after the first midterm
- Next Monday is a holiday - no 61A events happening (lecture, lab, OH)
    - More logistics will be announced on Ed once they are finalized
    - Enjoy your long weekend :)
- Check out Ed for more announcements!
- When emailing me, please try to put sth like `[CS 61A]` in the title so it's easier for me to sort things out!

VALUES, EXPRESSIONS, AND STATEMENTS 🧮

EXPRESSIONS AND VALUES

- Think of expressions as "questions" and values as "answers" - Python takes your questions, thinks a bit, and then give syou answers.
- The act of evaluation (and in fact, the whole point of an interpreter) is to turn expressions into values
- Expressions
    - Expressions evaluate to values
    - Eg: call expressions, arithmetic expressions, boolean expressions
- Values
    - Values are the only thing we can assign to variables, return from functions, pass into functions, etc.
    - Eg: numbers, strings, lists, functions, objects, etc.

EXPRESSIONS VS. STATEMENTS

- Expressions
  - Every expression evalutes to a value
  - Some may have side effects that modify the state of our program
- Statements
  - Do not evaluate to values
  - They serve some purposes in modifying the state of our program
    - For example, `while` loops allow us to repeat blocks of code multiple times
  - Although `def` statements create function values, they don't *evaluate* to function values themselves - instead, they *create a local binding* that binds the function name to the function value, like an assignment statement
- Why do we care about this?
  - It means we cannot return or print an assignment statement, or any other statement
  - We'll learn some expressions (namely, lambdas) that looks like statements but are different in some key ways, so it's good to learn the difference early

CONTROL STRUCTURES 🕹️

CONDITIONAL STATEMENTS

`if-elif-else` Syntax

```
if <cond_expr_1>:
    <suite_1>
elif <cond_expr_2>:
    <suite_2>
else:
    <suite_3>
```

- Conditional expressions - essentially any expression can be a conditional expression, and it evaluates to some value that's either truthy or falsy
- There can be 0 or 1 `else`
- There can be ≥ 0 `elif`
- Only the first `if` or `elif` that evaluates to a truthy value will have its corresponding indented suite be executed
- If none of the conditional expressions evaluate to a truthy value, the `else` suite is executed.

BOOLEAN OPERATORS 🎏

TRUTHINESS AND FALSINESS

Truthy Values

- Treated as practically "true" in boolean contexts (if/while conditions, and/or/not expressions)
- Everything that's not falsy is truthy

Falsy Values

- Treated as practically "false" in boolean contexts (if/while conditions, and/or/not expressions)
- Including: `0`, `None`, `""` (empty string), `[]` (empty list), etc.
- See here for a comprehensive list

BOOLEAN OPERATORS

- Some arithmetic expressions evaluate to boolean values

```
>>> 6 + 1 == 7
True
>>> 8 > 9
False
```

- Python boolean operators: `not`, `and`, and `or`
  - Priority: `not` > `and` > `or`
  - Use parenthesis to make your code more readable!
- `not` - returns the opposite boolean value of the following expression
  - always return either `True` or `False`

```
>>> not 0
True
>>> not -1
False
```

SHORT CIRCUITING

- Short circuit - not every operand gets evaluated
- `and`
  - evalutes from left to right until the first FALSY value or the last value
  - return the last thing that's evaluated
- `or`
  - evalutes from left to right until the first TRUTHY value or the last value
  - return the last thing that's evaluated
- If an error occurs, the execution flow is terminated immediately

```
>>> True and 1 / 0 and 2
ZeroDivisionError
>>> True or 1 / 0 or 2
True
```

```
>>> 1 and 2 and 3
3
>>> 1 or 2 or 3
1
```

WORKSHEET!

EVALUATION RULES

📡

EVALUATION RULES

Function Calls

1. Evaluate the operators
2. Evaluate the operands
3. Apply the operator to the values from evaluating the operands
    1. Create a new frame, bind the formal parameters to the actual argument values
    2. Evaluate the body of the function in the context of this new frame

EVALUATION RULES

Assignment Statements

1. Evaluate all of the expressions on the RHS of the assignment operator (the single equal sign) from left to right
2. Bind all the names on the LHS to the resulting values in the current frame

* Names can only bind to values, not other names!

```
a, b = 1, 2
a, b = b, a
# In Python, you can swap the values of multiple variables in one line
a, b = b + 1, a + b
# If there are multiple arguments to print, they will be seperated
# by a whitespace in the outputted line
print("a =", a, ", b =", b)
# output: a = 3, b = 3
```

EVALUATION RULES

Variable Lookup

1. Look it up in the current frame
2. If not found, go look in the parent frame, and up and up until global. If it's not there, then error.

* Built-in functions like `max` and `min` are usually not displayed on the environment diagram.

```
a, b = 1, 2
a, b = b, a
# In Python, you can swap the values of multiple variables in one line
a, b = b + 1, a + b
# If there are multiple arguments to print, they will be seperated
# by a whitespace in the outputted line
print("a =", a, ", b =", b)
# output: a = 3, b = 3
```

ATTENDANCE! 🤠

go.cs61a.org/mingxiao-att

- Also linked on our section website!
- Slides: go.cs61a.org/mingxiao-index