Final Exam Topics List

May 11, 2022

List of Material Covered in class and labs

- Lecture 1: slides 14-21 for decimal, binary and hex.
- Data Representation Examples: Additional Examples on Binary and Hex conversion
- Lecture 2 - Python 3 Basics
- Lectures 3-9
- Lectures 10 - Files slides 9, 11-19
- Lecture 11 - Formatted printing: Basic conversion types ('d', 'f', 's'), the width, and the precision (for floats) options.See examples below.
- Lecture 12 - Modules: Import statement styles: slides 2-7, 17-18, 25 (example), 33 only.
- Lecture 13 - More Functions
- Lectures 14-15: Object Oriented Programming
- Lecture 16: Exceptions
- Labs 1-13

Tokens

- Recognize and know how to use these reserved words:

  `and`
  `as`
  `break`
  `class`
  `continue`
  `def`
  `del`
  `elif`
  `else`
  `except`
  `False`
  `finally`
  `for`
  `from`
  `if`
  `import`
  `in`
  `is`
  `None`
  `not`
  `or`
  `raise`
  `return`
  `True`
  `try`
  `while`

- `int` - Decimal strings of digits *only*.
- `float` - Decimal digits with decimal point and optional integer exponent, e.g. `6.022e+23`, `1.`, `6.626e-34`, `.95`
- `str` - Single, double, and triple quoted. Know the significance of the sequence `'\n'` in a string.
- Lists - Lists of integers, floats, strings, mixed lists, 2-dimensional lists

Expressions

- Binary math operators: addition (+), subtraction (-), multiplication (*), remainder (%), floor division (//), real division (/), and exponentiation (**).
- Unary math operator: negation (-).
- Basic operator precedence: exponentiation is evaluated first (right-associative), then multiplication/division/remainder, then addition/subtraction (left-associative).
- Parentheses.
- Rule for promotion of `int` to `float`.
- Comparison operators: ==, !=, <, >, <=, >=. result is `True` or `False`.
- Boolean operators: `and`, `not`, `or`.
- Other operators: `is`, `in`.
- Index expressions: `lst[0]`, `lst[-1]`, `string[j-1]`
- Slice expressions: `lst[i:j + 1]`, `string[1:]`, `lst[::-1]`. A slice of a given type always returns an object of the same type as the original object.
- Function calls, e.g. `max(a, b)`, `min(a, b)`, `len(x)`, `sum(lst)`, `print(x, y, z, file=fp)`

Functions

- Default argument values: def f(x, y=1, z=2):
- Keyword arguments:
  `f(3)=f(x=3)=f(3,y=1,z=2)=f(3,1,z=2)=f(3,1,2)`
- Keyword arguments Example:
  print(x, end=' ', file=fp)
- Recursion vs. iteration.

Statements

- Assignment and short assignment (e.g. +=, *=)
- Expression (e.g. call to `print()`)
- `break` - Exit enclosing loop.
- `continue` - Skip to next iteration of loop.
- `for item in iterable`: Repeat for every item in iterable.
- `for ind in range(len(iterable))`: Repeat for every index in iterable.
- `Rules to decide which for loop` to use.
- `def` - Parameters, locals, default values.
- `if/elif/else` - Choose one of several actions.
- `import` - with or without `from`, different ways
- `try` - Handles exceptions. Comes with `except` clause(s), an optional `else` clause. Understand what happens when an exception occurs inside a `try` statement.
- `raise` - Changes program flow by signaling that an exception has occurred.
- `while` - If a condition is true, repeat statements.
- nested `while` and nested `for`

Types

- `bool` - `True` or `False`. Conversion of other types to boolean values, rules for results of boolean expressions.
- `float` - A floating-point number, `float()` will convert a `str` or `int` to a `float`.
- `int` - An integer, `int()` will convert a `str` or `float` to an `int`.
- `list` - A mutable sequence (or array) of values. A literal list is a comma-separated sequence of expressions surrounded by square brackets. Use of operators for comparison >, <, >=, <=, ==, !=. Concatenation with +, repetition with *. Important methods:

`append()`
`clear()`
`copy()`
`count()`
`extend()`
`index()`
`insert()`
`pop()`
`remove()`
`sort()`

- `str` - Immutable text. `str()` will convert any type to a string for output. Important methods:
  - `format()` - Basics of width, precision, and conversion types `'d'`, `'f'`, and `'s'`.
  - `split()`, `join()` - Convert to or from a list of strings.
  - `index()`, `rindex()`, `find()`, `rfind()` - Search for a substring. Know the difference.
  - `upper()`, `lower()`: convert to upper or lower case.
  - `isupper()`, `islower()`, `isalpha()`, `isdigit()`, `isalnum()`
  - Comparison rules, e.g. know why these are both `True`:

    ```
    "Apple" != "apple", "grape" < "grapefruit"
    ```

- `tuple` - An immutable list. Literal tuples are normally enclosed inside parentheses. Single-element tuple requires trailing comma. Operators same as with `list`. Methods: `index()` and `count()`

- `dict` - Associates values with a set of keys. Literal dictionaries are created by enclosing a list of key-value pairs in curly braces:

  ```
  d = { k1 : v1, k2 : v2, k3 : v3 } # check notes.
  x = d[k3]    # x now equal to v3
  d[k3] += 1   # increment the value associated with k3
  ```

  Dictionaries are *mutable*. Reading from a nonexistent key using square brackets raises an error. Dictionaries do not support the *, +, >, >=, <, or <= operators. The key can be any immutable type; the value can be *any* type. Important methods:

  `clear()`
  `copy()`
  `get()`
  `items()`
  `keys()`
  `values()`
  `setdefault()`

- Understand the term *iterable* as a shorthand for `list`, `str`, `dict`, or `tuple` types.

- `object` - The mutable type used as the base class for all Python classes.

- `Exception` - The base class for most Python exceptions. Know a few of the common exception classes: `ValueError`, `ZeroDivisionError`, `IndexError`, `NameError`.

- Mutable (`list`, `dict` and `objects`) vs. immutable (all other types so far).

Object-oriented programming

- Write class methods or main program code for a short `class` similar to those for `point` or `Student`.

- Know how to define and call a method on an object (instance).

- Recognize (and implement) the following special method names where obj1 and obj2 are instances of class_name:
  - `__init__` : Implicitly called as, `obj1=class_name(parameters)`
  - `__repr__` : Implicitly called as, `print(obj1)`
  - `__eq__` : Implicitly called as, `obj1 == obj2`
  - `__add__`: Implicitly called as, `obj1 + obj2`
  - `__mul__`: Implicitly called as, `obj1 * obj2`

Formatted printing

- The `format` method of `str` object is called, and it returns a string.
- Use of integer index (before colon) to select argument.

```
>>> a,b,c = 'x','y','z'
>>> print('{2} {2} {1}'.format(a,b,c)) # 2 is c and 1 is b
>>> z z y
```

- Know the 's', 'd', and 'f' format conversions.
- Know the use of width and precision (for floats).

```
>>> '{:10s}'.format('hello')
'hello     '
>>> '{:.3f} {:.4f}'.format(3.141592)
'3.142 3.1416'
>>> '{:3d} out of {:3d}'.format(9, 10)
'  9 out of  10'  #integers right-aligned by default
```

Builtin functions

`abs()`
`all()`
`any()`
`enumerate()`
`input()`
`len()`
`max()`
`min()`
`open()`
`range()`
`round()`
`sorted()`
`sum()`
`type()`
`zip()`

User-defined functions

- parameters and arguments
- calling a function
- return statement or statements
- Recursion vs. iteration.

Algorithms

- Use of a list of lists to represent a simple 2-dimensional matrix and how to access matrix elements.
- Basic recursion and the recursive form of simple algorithms such as factorial, list reversal, or list sum.

Files

- Files are accessed using `open()`

```
fp = open("data.txt")
```

- File are closed using `close()`

```
fp.close()
```

- Reading bytes (characters) from a file:

```
#File is one block or one line only.
text = fp.read(100) # reads 100 bytes
text - fp.read() # reads all
```

- Write to a file using print(..., file=fp)
- Reading from a text file line-by-line:

```
fp=open("data.txt")
for line in fp:  #for line in fp.readlines():
# Another way
while True:
   line = fp.readline() #read 1 line
```