15-110 Quiz1 Notes Sheet

Algorithms & Abstraction

Algorithms: procedures that specify how to do a task or solve a problem
Abstraction: changing the level of detail used to represent/interact with a system

Designing algorithms:
Little abstraction: assume no prior knowledge, need to define everything
Moderate abstraction: assume user has some basic knowledge already
Heavy abstraction: can make a lot more assumptions about incoming knowledge

Programming Basics

Integer (`int`): whole numbers (`14`)
Floating point number (`float`): numbers with a fractional part (`5.735`)
String (`str`): text in quotes (`"Sup all"`)
Boolean (`bool`): truth value (`True`)

Number operations: `+`, `-`, `*`, `/`, `**`
Text operations: `+`
Comparison ops: `<`, `>`, `<=`, `>=`, `==`, `!=`

Expression: code that evaluates to a data value
Statement: code that can change the state of the program
Variable assignment: `x = expr` stores the value of `expr` in the variable `x`
Variables: `x` evaluates to the value stored in the variable `x`

When dealing with an error:

1. Look for the line number
2. Look at the error type
3. For SyntaxErrors, look for the inline arrow
4. For other errors, read the error message

Data Representation

Number system: a way of representing a number using symbols. Currency, decimal, etc

Binary numbers: numbers in the base 2 system, composed of 0s and 1s.
Bit: a single digit in binary
Byte: eight bits interpreted together

Translate binary to decimal: add together the powers of 2 represented by the 1s. The first eight powers of 2 are 1, 2, 4, 8, 16, 32, 64, and 128.

Translate decimal to binary: repeatedly look for the largest power of 2 that fits in the decimal and remove it

Interpret binary as color: represent a single color with RGB (Red-Green-Blue). Each color component is represented by three bytes- intensity of red, then green, then blue.

Interpret binary as text: make a lookup table (like ASCII) that maps characters to numbers. Convert each byte to a number and look it up in the table.

15-110 Quiz1 Notes Sheet

Function Calls

Function: an algorithm implemented abstractly in Python that can be called on specific inputs

Arguments: input values to function call
Returned value: evaluated result, the output. If no output, defaults to `None`
Side effect: visible things that happen as the function runs (printing, graphics, etc)

`print(expr)` - show `expr` in interpreter
`abs(num)` - absolute value of `num`
`pow(x, y)` - raises `x` to power of `y`
`round(x, y)` - round `x` to `y` sig. digits
`type(expr)` - type of evaluated `expr`

Library: a collection of functions that need to be imported to be used

```
import libraryName
```

`math.ceil(x)` - ceiling of `x`
`math.log(x, y)` - log of `x` with base `y`
`math.radians(x)` - degrees to radians
`math.pi` - pi (to some number of digits)

`random.randint(x, y)` - random int in range [`x`, `y`]
`random.random()` - random float in range [0, 1)

```
canvas.create_rectangle(a,b,c,d)
```
- draw a rectangle from point (`a`, `b`) to point (`c`, `d`)

```
canvas.create_rectangle(a,b,c,d,
                        fill="blue")
```
- fill in the rectangle with the color blue

Function Definitions

Function definition: abstract implementation of an algorithm. Provides input with *parameters* (abstract variables), produces a result with a *return statement*.

```
def funName(args):
    # body
    return result
```

Local scope: variables in function definitions (including parameters) are only accessible within that function.

Global scope: variables at the global (top) level are accessible at the top-level, and by any function.

Function Call Tracing: Python keeps track of the functions it is currently calling in nested function calls. When Python reaches a return statement, it returns the value to the most recent function that called the current function.