

15-110 Exam1 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: +, *, in

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 Exam1 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`

`input(msg)` - accepts user input

`ord(c)` - ASCII value of `c`

`chr(x)` - character of ASCII value `x`

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.

15-110 Exam1 Notes Sheet

Booleans, Conditionals, & Errors

Logical operators: `and`, `or`, `not`

Short circuit evaluation: Python only evaluates the second half of a logical operation if it needs to

Conditional statement: control structure that allows you to make choices in a program.

```
if booleanExpr:
    ifBody
elif booleanExpr:
    elifBody
else:
    elseBody
```

Syntax Error: an error that occurs when Python cannot tokenize or structure code. Examples: `SyntaxError`, `IndentationError`, `Incomplete Error`

Runtime Error: an error that occurs when Python encounters a problem while running code. Examples: `NameError`, `TypeError`, `ZeroDivisionError`

Logical Error: an error that occurs when code runs properly but does not produce the intended result. Often (but not always) caused by a failed test case with `AssertionError`

```
assert(funName(input) == output)
```

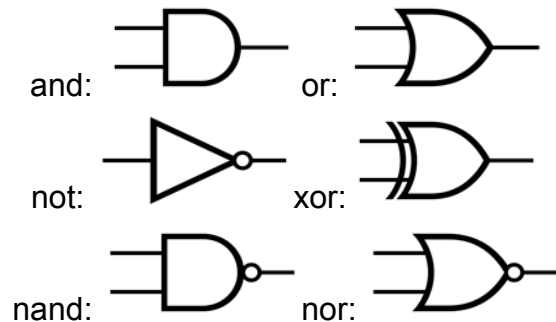
Circuits and Gates

Circuit: a hardware component that manipulates bits to compute an algorithmic result. Can also be simulated with an abstract version.

Gate: an abstract component of a circuit. Takes some number of bits as input and outputs a bit.

Gates: \wedge (and), \vee (or), \neg (not), \oplus (xor); also nand and nor (no special symbols)

Gates (in circuits):



Truth table: a table that lists all possible input bit combinations and the resulting output for a particular gate or circuit

Half-adder: a circuit that takes two one-digit binary numbers, adds them, and outputs two digits as the result

Full adder: a circuit that takes two one-digit binary numbers and a carried-in digit, adds all three, and outputs two digits as the result

N-bit adder: a circuit that takes two n-bit numbers, adds them together by chaining together n full adders, and outputs a n+1-digit result

15-110 Exam1 Notes Sheet

While Loops

While loop: a control structure that lets you repeat actions while a given Boolean expression is `True`

```
while booleanExpr:  
    whileBody
```

Infinite loop: a while loop that never exits due to the state of the program

Loop control variable: a variable used to manipulate the number of times a loop iterates. Requires a start value, update action, and continuing condition.

For Loops

For loop: a control structure that lets you repeat actions a specific number of times

```
for var in range(rangeArgs):  
    forBody
```

Range: a function that generates values for the loop control variable in a for loop. Can take 1-3 inputs.

```
range(end) # [0, end)  
range(start, end) # [start, end)  
range(start, end, step)  
# step provides the increment
```

Strings

Index: access a specific value in a sequence based on its position. Positions start at `0` and end at `len(seq)-1`. Non-existent indexes result in `IndexError`.

```
strExpr[index]
```

Slice: access a subsequence of a larger sequence based on a given start, end (not inclusive), and step

```
strExpr[start:end:step] # slice  
strExpr[start:end] # also slice  
# default to 0:len(strExpr):1
```

Looping over strings: use range and indexing to access one character at a time.

```
for i in range(len(strExpr)):  
    something with strExpr[i]
```

General Control Structures

Control flow chart: chart that designates how a program steps through commands. Uses branches for conditional checks and arrows leading back to previous commands for loops.

Nesting: a control structure can be included in the body of another control structure through use of indentation.

Nested loop: a loop with another loop in its body. The inner loop is fully executed for each iteration of the outer loop.