CS Scholars Review 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`)
Boolean (`bool`): truth value (`True`)
String (`str`): text in quotes (`"Sup all"`)
List (`list`): ordered collection of data values (`[1, 'a']`)

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`
Augmented assignment: shorthand to update a variable in place; `x += 1`

Errors, Debugging, and Testing

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)
```

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 Runtime Errors, read the error message
5. For Logical Errors, run the function call to get the actual output

Debugging Strategies: rubber duck debugging, printing and experimenting, thorough tracing

Test Case: a line of code that tests whether a function when called on a specific input returns the correct output. Test normal, large, edge, and special cases, and produce varying outputs.

CS Scholars Review 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)

Built-in Functions:
`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)` - turns user input into string

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`). Use `canvas.create_oval` to draw an oval & `canvas.create_line` to draw a line with similar coordinates.
`canvas.create_polygon(a,b,c,d,e,f)` - draw a polygon using the (x,y) points
`canvas.create_text(a,b,text=s)` - draw the text in `s` at `(a,b)`
`canvas.create_image(a,b,file=f)` - draw the image store in `f` at `(a,b)`

Keyword argument: an argument that can be included or can be left out and set to a default value. Tkinter examples: fill, width, font, anchor

```
canvas.create_rectangle(a,b,c,d,
                        fill="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.

CS Scholars Review 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
```

Loops

For loop: a control structure that lets you repeat actions a specific number of times, or over a specific data structure.

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

for var in sequenceValue:
    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
```

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.

Nesting and Top-Down Design

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

Nested conditionals: when two conditionals are nested, both must evaluate to True to reach the inner body

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

Nesting in functions: when a return statement is reached in a nested structure, the function immediately exits.

Helper function: a function that helps solve a big problem by solving a subpart of the problem.

Top-down design: solve a complicated problem by breaking it into several smaller problems and solving separately

CS Scholars Review Notes Sheet

Strings and Lists

Membership: can check if an item exists in a sequence or not

```
value in sequence
```

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`.

```
seqExpr[index]
```

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

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

Looping over sequences: use range and indexing to access one value at a time.

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

Method: a function called directly on a data value

```
result = value.method(args)
```

Methods:
`s.isdigit()`/`s.islower()`/
`s.isupper()` - checks that property of `s`

`L.count(item)` - # times `item` appears

`L.index(x)` - index of `x`, error if missing

`s.lower()`/`s.upper()` - makes new version of `s` that is lowercase/uppercase

`s.replace(a, b)` - new version of `s` with `a` replaced by `b`

`s.split(delim)` - makes a list of parts of `s` separated by `delim`

Destructive Method: a method that modifies the value it is called on directly instead of returning a new result

```
value.method(args) # no assign
```

Destructive Methods:
`L.append(val)` - adds `val` to end

`L.remove(val)` - removes `val` from `L`

`L.sort()` - sorts `L`

CS Scholars Review Notes Sheet

User Interaction

Text-based Interaction: create an interactive program loop by asking the user for input with `input`, using `print` to display output, and looping with `while` until some condition is met.

Input Validation: ensure that user input matches requirements, and force them to type the input again if it doesn't.

Event-Based Interaction: create an interactive program loop by receiving input from mouse and keyboard and displaying output as graphics.

MVC (Model-View-Controller): an interaction framework where functions work in tandem using a shared data structure instead of running sequentially. Store components in the *model*; update graphics from the *view*; call rule functions from the *controllers*.

```
# set up initial model
# data.var = value
makeModel(data)

# display current model
# use data.var in canvas call
makeView(data, canvas)

# update data.var on key event
# check event.char, event.keysym
keyPressed(data, event)

# update data.var on mouse event
# check event.x, event.y
mousePressed(data, event)
```

Real-World Coding

Style: the decisions you make while coding about how to organize and implement algorithms

Clarity Principles: to write code that is easy to read, use consistent formatting, use good naming conventions, don't include unnecessary code, and remember to document.

Robustness Principles: to write code that will be easy to modify later on, avoid repetitive code, avoid magic numbers, join up related conditionals, and test all functions.

External library: a library outside of the main Python language that can be installed into Python.

Documentation: instructions on how to use a library available online. Describes existing functions and what they do.

Install modules with:

```
pip install name
```