CIS 231

Python – Ch. 5

Conditionals and recursion

5.1 Floor Division and Modulus

- Floor division - truncates the quotient
  - e.g. `5 // 2` returns 2
- Modulus - Division operator that returns the remainder rather than the quotient
  - e.g. `x % y` returns the remainder when x is divided by y –> `5 % 2` returns 1
- Both work with `int`s and `float`s
- `%` - Handy for divisibility, digit extraction, and other uses
- Always be aware of potential divides by zero

5.2 Boolean expressions

- Evaluate to either `True` or `False`
  - case-sensitive – must be capitalized
- *Relational operators* compare two values and will return a boolean result:

```
# == is a relational operator
   (equals), rest are on p.40
x == 0
```

- These boolean results will be used for conditional execution and can also be saved as variables

5.3 Logical operators

- Operators that use boolean operands and allow you to combine multiple relations
- `and`, `or`, `not`

```
# is x both positive and odd?
posOdd = x > 0 and x % 2 == 1
```

- Non-zero numbers evaluate as `True`, so you could do this…(note the type of answer)

```
posOdd = x > 0 and x % 2
```

5.4 Conditional execution

- Allows us to have some statements only executed when a certain condition(s) is True
- We designate code as being conditional by placing it inside an `if` statement
- The statement(s) impacted by the condition are indented

```
if x > 0:
   print(“x is positive”)
```

5.5 Alternative execution

- You may want something to happen every time you come to that part of the program, but do one of two possible things
- Add an `else` clause to specify what will be done if the condition is False

```
if x % 2 == 0:
   print(“x is even”)
else:
   print(“x is odd”)
```

5.6 Chained conditionals

- Sometimes you may have more than two possible paths of execution – chained conditionals allow you to add any number of `elif` (short for `else if`) clauses to check other conditions
- If will proceed through the structure until if finds a match, then skips the rest of the conditions and continues with the program
- Code example on next slide

Chained conditional code

```
if x < y:
  print(“x is less than y”)
elif x > y:
  print(“x is greater than y”)
else:
  print(“x and y are equal”)
```

5.7 Nested conditionals

- An alternative to chained conditionals
- Include (or *nest*) an `if-else` inside another `if` or `else` clause
- Typically used in many other languages
- Many of the use cases for these are handled by `elif`

5.8 Recursion

- A function calling itself
- Useful when a solution is not in hand but we can have the function solve a subset of the current problem
- Typically used when a loop is not practical
- Will be used for more advanced topics down the road
- Almost always best to use a loop if you can

5.9 Stacks - recursion

- In recursion the idea is to work to a base case
- A base case is one where the answer is now known and there is no more need for the function to call itself
- A recursive (or make progress) case is one that requires the function to call itself
- As with other functions, when it calls itself the calling function is saved on the stack (more)

Stacks – recursion cont.

- Therefore when a function calls itself, all of the calls are preserved even though the same code is being executed more than once
- It keeps track of where you were in each call and what the variables are
- Figure 5-1 on p. 44

5.10 Infinite recursion

- If you don’t have a base case or a way to move towards one, the function will keep calling itself until the maximum depth has been exceeded
- This means that the calls exceeded the allocated amount of stack space, which is commonly known as a *stack overflow*
- Sooo…always have at least one base case

5.11 Keyboard input

- So far we've had to change values manually ("hard coding") in order to work with different values
- To be truly useful we need to be able to bring in (input) values from outside the program
- The most common way to do that is the user inputting values via the `input()` function
- Code example on next slide

Keyboard input example

```
whatWasTyped = input()
print(whatWasTyped)
# adding a prompt for the user
name = input("What's your name?\n")
print(name)
```

- Notice that the `\n` (*newline*) send the cursor down to the next line – you can omit it to have the input happen next to the prompt

Up Next

- Ch. 6 – Fruitful Functions
  - Return values
  - Incremental development
  - Composition
  - Boolean functions
  - More recursion
  - Leap of Faith
  - One more example
  - Checking types