**MITOPENCOURSEWARE:**

Lecture 1: what is computation?

Diagram

Description automatically generatedLO

* Course info
* What is computation
* Python basics
* Mathematical operations
* Python variables and types

**Topics:**

* Represent knowledge with data structures
* Iteration and recursion as computational metaphors
* Abstraction of procedures and data types
* Organize and modularize systems using object classes and methods
* Different classes of algorithms, searching and sorting
* Complexity of algorithms

What does a computer do?

Fundamentally it performs calculations/ a billion calculations per second. It also remembers results 100s of gigabytes of storage

What kind of calculations? Built-in to the language. Ones that you define as a programmer.

computers only know what you tell them!

Types of knowledge:

Declarative knowledge is statement of fact – someone will win a prize

Imperative knowledge is a recipe or how-to.

1. Students sign up at : <http://bit.ly/60001-raffle>
2. Ana opens her IDE
3. Ana chooses a random number between 1st and n responder
4. Ana finds the number in the responder’s sheet. Winner!

Python random number code: random.randint(n,n)

Numerical example:

* Square root of a number x is y such that y\*y = x
* Recipe for deducing square root of a number x (16)

1)start with a guess, g

2)if g\*g is close enough to x, stop and say g is the answer

3)otherwise make a new guess by averaging g and x/g

4)using the new guess, repeat process until close enough

Table

Description automatically generated

Diagram

Description automatically generatedWhat is a recipe:

1)sequence of simple steps

2)flow of control process that specifies when each step is executed

3)a means of determining when to stop 1+2+3 = an algorithm

Computers are machines:

* How to capture a recipe in a mechanical process
* Fixed program computer – calculator
* Stored program computer – machine stores and executes instructions

Stored program computer:

* Sequence of instructions stored inside a computer:
* Built from predefined set of primitive instructions

1. Arithmetic and logic
2. Simple tests
3. Moving date

* Special program (interpreter) executes each instruction in order
* Use tests to change flow of control through a sequence
* Stop when done

Basic primitives

* Turing showed that you can compute anything using 6 primitives
* Modern programming languages have more convenient set of primitives
* Can abstract methods to create new primitives
* Anything computable in one language is computable in any other programming language

Creating recipes:

* A programming language provides a set of primitive operations
* Expressions are complex but legal combinations of primitives in a programming language
* Expressions and computations have values and meanings in a programming language

Aspects of languages

* Primitive constructs:
* English: words
* Programming language: numbers, strings, simple operators.
* Syntax: English: “cat dog boy” – not syntactically valid

“cat hugs boy” – syntactically valid

* Programming language: “hi” 5 – not syntactically valid

3.2\*5 – syntactically valid

English: “I are hungry” – syntactically valid but static semantic error

Programming language: 3.2\*5 – syntactically valid

3+”hi” – static semantic error

Semantics: is the meaning associated with a syntactically correct string of symbols with no static semantic errors – English: can have many meanings “Flying planes can be dangerous”

Programming languages: have only one meaning but may not be what programmer intended

Where things go wrong:

* Syntactic errors: common and easily caught
* Static semantic errors: some languages check for these before running program. Can cause unpredictable behaviour
* No semantic errors but different meaning than what programmer intended: program crashes, stop running. Program runs forever. Program gives an answer but different than expected.

Python programs:

* A program is a sequence of definitions and commands.
* Definitions evaluated
* Commands executed by python interpreter in a shell
* Commands (statements) instruct interpreter to do something
* Can be typed directly in a shell or stored in a file that is read into the shell and evaluated
* Problem set 0 will introduce you to these in anaconda

Objects:

* Programs manipulate data objects
* Objects have a type that defines the kind of things programs can do:
* Ana is a human so she can walk, speak English, etc
* Chewbacca is a wookie so he can walk, “mwaaarhrhh”, etc
* Objects are:
* Scalar (cannot be subdivided)
* Non-scalar (have internal structure that can be accessed)

Scalar objects:

* Int – represents integers, ex. 5
* Float – represent real numbers, ex.3.27
* Bool – represent Boolean values true and false
* Nonetype – special and has one value, none
* Can use type() to see the type of an object
* >>> type (5)

Int

>>> type (3.0)

Float

Type of conversions (cast)

* Can convert object of one type to another
* Float (3) – converts integer to float 3.0
* Int (3.9) truncates float 3.9 to integer 3

Printing to console:

* To show output from code to a user, use ‘print’ command

In [11]: 3+2

Out [11]: 5

In [12]: print (3+2)

5

Expressions:

* Combine objects and operators to form expressions
* An expression has a value, which has a type
* Syntax for simple expression

<object> <operator> <object>

Operators ON ints and floats:

* I+j – the sum
* I-j – the difference if both are ints, result is int. if either or both are floats, = float
* I\*j – the product
* i/j – division – float
* i%j – the remainder when I is divided by j
* i\*\*j – I to the power of j

Binding variables and values

* equal sign is an assignment of a value to a variable name
* pi = 3.14159
* pi\_approx = 22/7
* value stored in computer memory
* an assignment binds name to value
* retrieve value associated with name or variable by invoking the name, by typing pi

Abstracting expressions:

* why gives names to values of expressions?
* To reusue names instead of values
* Easier to change code later
* Pi = 3.14159
* Radius = 2.2
* Area = pi\*(radius\*\*2)

Programming vs Math:

* In programming, you do not “solve for x”

Pi = 3.14159

Diagram

Description automatically generatedText

Description automatically generatedRadius = 2.2

# area of circle

Area = pi\*(radius\*\*2)

Radius = radius+1

Changing bindings:

* Can re-bind variable names using new assignment statements
* Previous value may still stored in memory but lost the handle for it
* Value for area does not change until you tell the computer to do the calculation again

**Lecture 2 – Branching and iteration**

Strings:

* Letters, special characters, spaces, digits
* Enclose in quotation marks or single quotes ( hi = “hello there”
* Concantenate strings (= put the strings together) ( name = “ana” / greet = hi + name / greeting = hi + “ “ + name
* Do some operations on a string as defined in python docs ( silly = hi + “ “ + name \* 3)

Input/output: print:

* Used to output stuff to console
* Keyword is “print”
* X = 1
* Print (x)
* X\_str = str(x)
* Print (“may fav num is” , x, “.”. “x=”, x)
* Print(“may fav num is “ + x+str + “. “ + “x = “ + x\_str)
* If you use a comma (,) python adds a space

Input/output: input(“”)

* Prints whatever is in the quotes
* User types in something and hits enter
* Binds that value to a variable – text = input(“type anything… “) . print 5(text)
* Input gives you a string so must cast if working with numbers
* Num – int(input(“type a number….”)) // print (5\*num)

Comparison operators on int, float, string

* I and j are variable names
* Comparisons below evaluate to a Boolean
* I > j
* I>= j
* I < j
* I <= j
* I == j – equality test, true if I is the same as j
* I != j – inequality test, true if I not the same as j

Table

Description automatically generatedLogin operations on bools

* A and b are variable names (with Boolean values)
* Not a – true if a is false / false if a is true
* A and b – true if both are true
* A or b – true if either or both are true

Letter

Description automatically generated(based of the alphabet not what you assign to a/b e.g a = 7 b =6 / a > b = true)

(20.00 – video)

**Code Academy:**

Python:

Course 1:

Overview - In this unit, we will cover fundamental rules of probability including how to describe random events. We will cover topics such as set theory, conditional probability, joint probability, Bayes rule, probability distributions, and sampling distributions. These concepts are important in order to understand the likelihood of events, fit machine learning models, and perform hypothesis tests.

Probability is a way to quantify uncertainty. When we flip a fair coin, we say that there is a 50 percent chance (probability = 0.5) of it coming up tails. This means that if we flip INFINITELY many fair coins, half of them will come up tails. Similarly, when we roll a six-sided die, we say there is a 1 in 6 chance of rolling a five. What if we flip a coin in one hand and roll a die in the other at the same time. What is the probability that the coin comes up tails AND the die comes up as a five? Is there a way to quantify the probability that these two different events BOTH occur? In this lesson, we will walk through different rules of probability that help us quantify the probability of multiple random events.