* Course Overview
* Overview
  + Install python
  + Write basic code
* Installing python
  + Install on windows
    - Python.org
    - Download
    - Add python 3.7 to path variable
    - Use windows powershell
    - Type ‘python’
  + Installation on macos
    - Python.org
    - Download for mac os
  + Installation on linux
    - Default to recent version of ubuntu
* Interactive Python
  + REPL(read evaluate print loop)
  + Type ‘\_’ to refer to most recently printed value
    - Can be used in expressions
    - Only in REPL
  + Type ‘print(‘string’) to print to command line, new line per print
  + In python 2 type ‘print “string”’
    - Is a function call
  + In python 3 type ‘print(‘string’)
  + Type ‘exit()’ to exit REPL
  + Type ‘Ctrl-Z + Enter’ to exit REPL on windows
  + Type ‘Ctrl-D’ to exit REPL on linux/macOS
* Significant Whitespace
  + Control flow section are terminated by :, which requires a body
  + Ex)
    - for i in range(5):
    - x = i \* 10
    - print(x)
  + Leading whitespace is syntactically significant
  + Python uses indentation levels to demarked code blocks
    - Four spaces for each level
    - Use empty line to terminated block
  + range(num) is from 0 to num - 1
  + requires readable code
  + no clutter
  + human and computer can’t get out of sync
  + Whitespace rules
    - Prefer four spaces
    - Spaces over tabs
    - Never mix spaces and tabs
    - Be consistent on consecutive lines
    - Only deviate to improve readability
* Python Culture
  + In REPL type ‘import this’ to show zen of python
* Using the Standard Library
  + Use the import keyword
  + Ex)
    - import [module]
  + use ‘module\_name.attribute\_name’ to access attribute
  + ex)
    - math.sqrt(81)
  + type help(object) to see attributes
  + ex)
    - help(math)
  + they ‘q’ to return to REPL
  + type help(math.factorial) to gain more info function
  + type ‘from math import factorial’ to just bring in the function
  + ex usage) factorial(n)
  + can also rename the function
  + ex) from math import factorial as fac
    - fac(10)
  + / is floating point division
    - 5 / 2 return 2.5
  + // is integer division
    - 5 / 2 returns 2
  + len(string) to return count of char in string
  + str(other type) to convert to string
* Summary
* Overview
  + Scalar types
  + Relational operators
  + Flow-control
  + Scalar types
    - Integers, collections(dictionaries) etc
    - Int, flot, None(null object), True, False
  + int
    - unlimited precision signed integer
    - 0b[num] for binary
    - Ex) 0b10 -> 2
    - 0o[num] for octal
    - 0x[num] for hexadecimal
    - int(num) to convert num to integer
    - int(num, base) to convert to base integer
  + float
    - IEEE-754 double-precision with 53 bits of binary precision
    - 15-16 significant digits in decimal
    - [num]e[num of trailing zeros]
    - Ex) 3e8 = 300000000.0
    - float(num) to convert to float
  + nan: not a number
  + inf: positive infinite number
  + -inf: negative infinite number
  + int and float results in float
  + None: null value
    - Often represents the absence of a value
    - REPL never prints none results
    - Can be assigned to variables
  + Us ‘is’ to check for type, returns bool True or False
  + Ex) a is None
  + bool
    - Boolean logical values
    - True, False
    - 0 is falsey, every other numb is truthy
    - Empty list is falsey, non empty list is truthy
    - Empty string is falsey, non empty string is truthy
    - “False” string is truthy, “True” string is truthy
* Relational Operators
  + ==: value equality / equivalence
  + !=: value inequality / inequivalence
  + <: less-than
  + >: greater-than
  + <=: less-than or equal
  + >=: greater-than or equal
* Control Flow
  + Conditional statement branch execution based on the value of an expression
  + if expression:
  + block
  + elif:
  + block
  + else:
  + block
  + elif is else if
* While-loops
  + while expression:
  + block
  + Augmented assignment operators
    - -=
    - +=
    - \*=
    - Etc
  + break: many languages support a loop ending in a predicate test
    - do while loop
  + python requires you to use while True and break
  + break jumps out of the inner-most executing loop to the line immediately after it
  + input() takes input from user
* Summary
  + scalar types: int, float, None, bool
  + if-elif-else
  + Control-C generates a KeyboardInterrupt exception
* Overview
  + str, bytes, list, and dict
  + for-loop
  + collections types
    - str, bytes, list, dict,
* String
  + str: data type for strings in python
  + sequence of Unicode code points
  + Immutable: cant modify its content after creation
  + Strings are enclosed in ‘ ’ or “ “
    - You must be consistent
  + Use one or the other to escape ‘ or “
  + Ex) “It’s a good thing.”
  + Ex) ‘ “Yes!”, he said, “I agree!” ‘
* String Literals
  + Adjacent literal strings are concatenated by compiler to single string
  + Ex) “first” “second” -> ‘firstsecond’
  + String with Newlines
    - Multiline strings: spread the literal across multiple lines
    - Escape sequences: Embed escape sequences in a single-line literal
  + Multiline Strings: use “ “ “ or ‘ ‘ ‘
    - Use “””
    - Ex)
    - “”” This is
    - A multiline
    - String “””
  + Escape sequences: put ‘\n’ into strings
  + Ex) ‘This string\nspans multiple\nlines’
  + New lines and operating systesm
    - Windows: carriage-return, line-feed(\r\n)
    - Linux and macOS carriage-return(\r)
  + Pythons translate \n to the appropriate newline sequence for your platform
    - Universal newlines
  + Escape Sequences
    - “This is a \” in a string “ -> ‘This is a “ in a string’
    - ‘This is a \’ in a string’ -> “This is a ‘ in a string”
    - ‘This is a \” and a \’ in a string’ -> ‘This is a “ and a \’ in a string’
    - Use \\ to escape \
    - Ex)
    - Print(‘A \\ in a string’) -> A \ in a string
  + Raw strings: what you see is what you get
    - Prefix with r
    - Ex) prefix = r’C:\Users\Merlin\Documents\Spells’
  + str(parameter) to convert parameter to string
  + strings are sequence types
    - can use square brackets to access individual characters of string
    - ex) s = ‘parrot’
    - s[4] -> ‘o’
  + in python there is no separate character type, s[4] returns a string
  + type(parameter) to return type of parameter
  + help(str) to get methods and properties on string
  + str.capitalize() returns a new string with first character capitalized
  + ex)
    - c = ‘oslo’
    - c.captialize()
  + str is Unicode
  + python 3 source encoding is UTF-8
* Bytes
  + Data type for sequences of bytes
  + Raw binary data
  + Fixed-width single-byte encodings
  + Prefix with b
  + Ex)
    - b'data’
    - b”data”
  + support the same operations as string
  + bytes.split() split by be ‘delimiter’, empty for spaces
  + converting between strings and bytes
    - must know the encoding to turn string to bytes
    - decoding for byte to string
  + ex)
    - norks = “….”
    - data = norks.encode(‘utf8’)
    - norwegian = data.decode(‘utf8’)
* Lists
  + Sequences of objects
  + Mutable
  + A workhorse in python
  + [] is used to defined lists
  + Ex)
    - [1,9,8]
  + Use square brackets to access data
  + Can have different types of objects in the same list
  + list.append() to add to end of list
  + list() to use list constructor
* Dict
  + Fundamental data structure
  + Maps keys to values
  + Known as maps or associative arrays
  + {} to defined lists
  + { key: value}
  + Use [key] to access elements
  + If you use key that has not been added, a new entry will be added
  + Since python 3.7 entries are keep in insertion order
* For-loop
  + Visit each item in an iterable sequence
  + Ex)
    - for item in iterable:
  + body
  + For dictionary you get the keys
  + Ex)
    - for color in colors:
    - print(color, colors[color])
* Putting it all Together
  + Http data is provided by bytes
  + REPL is good for short-lived work and experimentation
  + Use an editor/IDE for larger or longer-lived projects
* Summary
* Overview
  + Reusable functions
  + Source code files called modules
  + Importing modules
  + Programs or scripts
  + Python execution model
* Modules
  + All python source file has extension .py
  + To run file type ‘python [filename].py’
  + To import into REPL
    - Type ‘python’ to enter REPL
    - Type ‘import [filename’
* Functions
  + Use def [function name](argurment list): to define a function
  + Ex)
    - def square(x):
    - return x \* x
  + \*\*: exponential operator
    - Ex) 2 \*\* 3 = 8
  + Naming special functions
    - \_ \_ feature \_ \_
  + Dunder
    - Our way of pronouncing special names
    - A portmanteau of “double underscore”
  + We’ll say “dunder name”
* \_\_name\_\_