**Fundamentals of Programming – Summary**

|  |  |  |
| --- | --- | --- |
| **Comments**  **#** character enables a comment | *# Comment on a single line*    name = "Alice" *# variable name has the value of Alice* | Week 1 |
| **print() Function**  The **print()** function prints a message to the console or output | *# Prints blank line*  print()  *# Prints message "Cheatsheet"* print("Cheatsheet") | Week 1 |
| **Synax** | Strings, Boolean, Lists, and so on | Week 1 |
| **Strings**  Strings contains characters enclosed by single quotes or double quotes | *# Single quotes* single = 'Single'  *# Double quotes* double = "Double" | Week 1 |
| **String Concatenation** The plus sign **+** can add two str types or Strings (and not just numeric types) | single = 'Single'  double = "Double"  *# String Concatenation*  *# concatenate becomes "SingleDouble"* concatenate = single + double | Week 1 |
| **Integer (and Float)**   * **Integer** must be a whole   number, it can be 0, positive or negative, but must be without decimals   * **Float** is a number, it can be positive or negative, but must contain one or more decimals | *# Integer or int*  height = 183  weather = -6  zero = 0  *# Float*  f = 1.33  *# If unsure, use type() function*  *# For example*  *# print type(f)* | Week 1 |
| **Variables**  Variables assign a value or store data using an equals sign **=** | *# Valid variable names and assignment* string\_var = "String is here"  int\_var = 10  float\_var = 10.13  bool\_statement = False  *# Variable value can be changed after* int\_var = 12 | Week 1 |
| **Arithmetic Operations**   * **+** for addition * **-** for subtraction * **\*** for multiplication * **/** for division * **%** for modulus (returns the remainder) * **\*\*** for exponentiation (number^number) | *# Arithmetic operations*    addition = 2 + 5 *# 7* subtraction = 30 - 40 *# -10* multiplication = 2 \* 90 *# 180* division = 4 / 4.0 *# 1.0* modulus = 13 % 2 *# 1* exponentiation = 2 \*\* 3 *# 8* | Week 1 |
| **Modulo operator (%)**  Modulo operation, with a percent sign **%**, returns the remainder of the division operation   1. 6 % 2 results 0 because 6 divided by 2 would return 3, leaving no remainder 2. 7 % 2 results 1 because 7 divided by 2 is not evenly divisible, leaving 1 remainder 3. 9 % 50000 results 9 because 9 divided by 50000 is not evenly divisible, leaving 9 remainder | *# Modulo operator %*  *#Example 1*  zero = 6 % 2 *# 0*    *#Example 2*  one = 7 % 2 *# 1*    *#Example 3*  nine = 9 % 50000 *# 9* | Week 1 |
| **Plus Equals**  A shortcut to update a variable value can be done by using the plus equals sign **+=** | *# Plus Equals (int)*  count = 0  count += 1 *# equivalent to count = count + 1*    *# Plus Equals (str)*  data = "Plus "  data += " Equals" *# equivalent to data = data + " Equals"* | Week 1 |
| **Errors**  PyCharm will (a) display the error type; and (b) information where the error has occurred | If unsure, use type() function  ^^^^^^  SyntaxError: invalid syntax | Week 1 |
| **NameError** NameError is produced variable(s) were not defined | *# NameError* print(result) | Week 1 |
| **SyntaxError**  SyntaxError is produced when Python syntax is not followed | *# SyntaxError*  result = 'Wrong quotes" | Week 1 |
| **ZeroDivisionError** ZeroDivisionError occurs when a number (denominator) is divided by a 0 or 0.0 | *#ZeroDivisionError*  numerator = 50  denominator = 0  result = numerator / denominator print(result) | Week 1 |
| **Boolean - False** | Empty, False, 0 None | Week 2 |
| **Boolean – True** | Any number (beside 0), True, String(that is not empty) | Week 2 |
| Equals and Not Equals | test = ""  print(bool(test)) *# False*  '13' == 13 *# False* | Week 2 |
| If Statement | day = 'Monday' *# Enter day here*  if day == 'Monday':  print('attend the Workshop') | Week 2 |
| Ask for User input | print('Enter your age: ')  age = input() *# stores String*  age = int(age) *# converts to int* | Week 2 |
| Logical Operators  1. and  2. or  3. not | |  |  |  | | --- | --- | --- | | First Condition | Second Condition | Output | | F | F | F | | F | T | F | | T | F | F | | T | T | **T** | | Week 2 |
| Logical operator: not | print(not True) *# prints False*  print(not False) *# prints True* | Week 2 |
| Else statement | credits = 35 *# Modify credits value here*  if credits >= 1000:  print("You are a Platinum member")  elif credits >= 500:  print("You are a Gold member")  elif credits >= 250:  print("You are a Silver member")  else:  print("You are a Red member") | Week 2 |
| Loops  (1) initialisation;  (2) repetitions;  (3) an ending condition | Iteration   1. Indefinite: where the number of times a loop is executed depends on the number of times a condition is met 2. Definite: where the number of times a loop will be executed is declared in-advance | Week 3 |
| Loops  typically utilised to iterate a  collection of items | grades = [35, 50, 65, 85]  print(grades[0]) *# 35*  print(grades[1]) *# 50*  print(grades[2]) *# 65*  print(grades[3]) *# 85* | Week 3 |
| For Loops   * a type of definite iteration * which will know how many times to iterate the loop because our grades list contains a collection of elements or items with a predefined length | for <temp var> in <collection>:  <action>  grades = [35, 50, 65, 85]  for g in grades:  *#Any code at this indentation*  *#will be executed on each*  *#iteration of the loop*  print(g)  *1. g is the <temp var>*  *2. grades list is the <collection>*  *3. print(g) is the <action> completed on every*  *iteration using the temporary variable g* | Week 3 |
| One line for loops | grades = [35, 50, 65, 85]  for g in grades: print(g) | Week 3 |
| Using Range in For Loops | for temp in range(11):  *#Iterates 11 times, from 0 to 10*  *#temp does nothing, but print*  *#statement repeats the message*  *#*  *range(11) = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]*  print('I must attend my workshops')  for temp in range(11):  print('Iteration number is: ' + str(temp))  *#Output:*  *#Iteration number is: 0*  *#Iteration number is: 1*  *#Iteration number is: 2*  *#Iteration number is: 3*  *#Iteration number is: 4*  *#Iteration number is: 5*  *#Iteration number is: 6*  *#Iteration number is: 7*  *#Iteration number is: 8*  *#Iteration number is: 9*  *#Iteration number is: 10* | Week 3 |
| While Loops | while <conditional statement>:  <action>  iteration = 0  while iteration < 11:  print('Iteration number is: ' + str(iteration))  iteration = iteration + 1  grades = [35, 50, 65, 85]  iteration = 0  while iteration < len(grades):  print(grades[iteration])  iteration = iteration + 1 | Week 3 |
| Loop Control  1. Break | grades = [35, 50, 65, 85]  for grade in grades:  pass  grades = [35, 50, 65, 85]  for grade in grades:  print(grade)  if grade < 50:  break  *# prints 35 because the break*  *# is after the print function* | Week 3 |
| Loop Control  2. Continue | grades = [35, 50, 65, 85]  for grade in grades:  if grade < 50:  continue  print(grade)  *# prints 50, 65, 85*  *# Order of "continue" before*  *# print function is important* | Week 3 |
| Loop Control  3. Else | grades = [35, 50, 65, 85]  for grade in grades:  print(grade)  else:  print('Completed the grades for loop.')  *"""*  *prints*  *35*  *50*  *65*  *85*  *Completed the for grades for loop.*  *"""* | Week 3 |
| Nested Loops | gradebook = [  ['Jing', 35],  ['Jim', 50],  ['Kerrie', 65],  ['Rakesh', 85]  ]  for student in gradebook  print(student)  for detail in student  print(details) | Week 3 |
| List Comprehension | *# Assuming if there is a <conditional statement>*  new\_list = [<expression> for <temp var> in <collection> <conditional statement>]  # If there isn't a <conditional statement>  new\_list = [<expression> for <temp var> in <collection>]  students = ['Jing', 'Jim', 'Kerrie', 'Rakesh']  new\_list = [name for name in students if 'i' in name]  *For example, creating a new\_list list with values from 0 to 10:*  new\_list = [i for i in range(11)]  *For example, creating a new\_list list with values from 0 to 10, but only including values 6 and above:*  new\_list = [i for i in range(11) if i > 5] | Week 3 |
| Summary of Python collections | 1. List: “is a collection which is **ordered** and **changeable(Mutable)**.  Allows duplicate members.”  2. Tuple: “is a collection which is **ordered** and **unchangeable**.  Allows duplicate members.”  3. Set: “is a collection which is **unordered**, **unchangeable**\*, and  unindexed. No duplicate members.”  4. Dictionary: “is a collection which is **ordered**\*\* and  **changeable(Mutable)**. No duplicate members.” | Week 4 |
| List   * list is created using square brackets [] * Each item is separated by a comma ,   + Good Python practice: insert a space after each comma * store a combination of different data types | list = ["String", 27, False, 25.65, [1, 2, 3, 4, "Count"], None]  *# Create an empty list*  list\_empty = []  list\_empty = list(()) | Week 4 |
| List Methods / Class | *# list\_name is our list*  *# .method() is a method of our list*  *# This example is for structure illustration,*  *# it will not work*  list\_name.method()  AttributeError: 'list' object has no attribute 'method'  *# Syntax for List methods*  list.method(parameter) | Week 4 |
| Reference  Check Methods - Documentation | Use the following official website:  1. Official Python Docs:  <https://docs.python.org/3/tutorial/datastructures.html>  • Use any other unofficial websites:  2. W3Schools Docs:  <https://www.w3schools.com/python/python_ref_list.asp>  3. GeeksforGeeks Docs:  <https://www.geeksforgeeks.org/list-methods-in-python/> | Week 4 |
| List Method: .append()  adds an element to the end of a list | grades = [35, 50, 65, 85]  grades.append(67)  print(grades) *# Prints [35, 50, 65, 85, 67]* | Week 4 |
| Adding multiple items to a list | grades = [35, 50, 65, 85, 67]  grades += [76, 78, 52] *# OR grades = grades + [76, 78, 52]*  grades.extend([68, 72]) *# square brackets are used here and above*  print(grades) *# prints [35, 50, 65, 85, 67, 76, 78, 52, 68, 72]* | Week 4 |
| Accessing list elements   * the first element in a list has an index of 0 * Negative index can be used to access last element(s) of a list | students = ['Jing', 'Jim', 'Kerrie', 'Rakesh']  print(students[1]) *# prints Jim*  print(students[-1]) *# prints Rakesh*   |  |  |  | | --- | --- | --- | | Element | Index | Negative Index | | 'Jing' | 0 | -4 | | 'Jim' | 1 | -3 | | 'Kerrie' | 2 | -2 | | 'Rakesh' | 3 | -1 | | Week 4 |
| Length   * len() function returns the number of items\* in a object\* | students = ['Jing', 'Jim', 'Kerrie', 'Rakesh']  print(len(students)) *# prints 4*  guess = 'Howmansafayhschatersarehere can you guess? :-)'  print(len(guess)) *# prints 46* | Week 4 |
| Remove list elements  1. grades.remove(value)  2. grades.pop(specific\_index)  3. del grades[specific\_index] | grades = [35, 50, 65, 85]  grades.remove(35)  print(grades) *# prints [50, 65, 85]*  grades = [35, 50, 65, 85]  grades.pop(0)  print(grades) *# prints [50, 65, 85]*  grades = [35, 50, 65, 85]  del grades[0]  print(grades) *# prints [50, 65, 85]* | Week 4 |
| Two Dimensional (2D) Lists  – Known as a nested list – this logic can be applied to create a three dimensional (3D) list | gradebook = [['Jing', 35], ['Jim', 50], ['Kerrie', 65], ['Rakesh', 85]]  gradebook = [  ['Jing', 35],  ['Jim', 50],  ['Kerrie', 65],  ['Rakesh', 85]  ]  print(gradebook[-1][1]) *# prints 85*  gradebook[0][1] = 35  gradebook[-4][1] = 35  gradebook[-4][-1] = 35 | Week 4 |
| Zip   * joins lists together. Extremely powerful! * If zip parameters – or iterators – lengths are different, than the iterator with the least element determines the length of the new iterator | students = ['Jing', 'Jim', 'Kerrie', 'Rakesh' , 'Candy']  grades = [35, 50, 65, 85]  gradebook\_zip = list(zip(students, grades))  *# print(gradebook\_zip) # zip creation - note the tuples*  *# [('Jing', 35), ('Jim', 50), ('Kerrie', 65), ('Rakesh', 85)]*  *# print(gradebook) # original*  *# [['Jing', 35], ['Jim', 50], ['Kerrie', 65], ['Rakesh', 85]]*  *#*  *"Candy" will be disappeared* | Week 4 |
| Tuples   * do the same operation as Lists * Tuples are unchangeable * However, instead declaring using square brackets * [] like Lists, it uses round brackets () | 1. .count(): returns the number of times the specified element appears in the list 2. .insert(): inserts the specified value at the specified position 3. .sort(): sorts the list ascending by   default   1. .sorted(): returns a sorted list of the specified iterable object | Week 4 |
| Built-in function | *# Syntax for a built-in function*  builtinfuncion(parameter) | Week 4 |
| Range | numbers = [0, 1, 2, 3, 4, 5] | Week 4 |
|  | 1. 0 to 5   range(6)   1. (2) 0 to 999   range(1000) | Week 4 |
| Reference  Range | W3Schools:  <https://www.w3schools.com/python/ref_func_range.asp>  even sequence of numbers ranging from -255 to 255  ans = list(range(-254, 256, 2))  print(list(range(-10, 10, 2)))  *# [-10, -8, -6, -4, -2, 0, 2, 4, 6, 8]*  print(list(range(-10, 10, 3)))  *# [-10, -7, -4, -1, 2, 5, 8]*  print(list(range(0, 10, 3)))  *# [0, 3, 6, 9]* | Week 4 |
| Slicing Lists | nums[start:end]  nums = [0, 1, 2, 3, 4, 5, 6, 7, 8]  len(nums) = 9  nums[2:7] *# For i = 2 ; i < 7*  nums[:n] *# For i = 0 ; i = n*  nums[:3] *# For i = 0 ; i = 3*  nums[n:] *# For i = len(nums)-n ; i < len(nums)*  nums[-6:] *# For i = len(nums)-6 ; i < len(nums)*  *# [3, 4, 5, 6, 7, 8]* | Week 4 |
| Sets  using curly brackets {}  1. Set items are unchangeable  a) The exception: add and remove items allowed  2. Set items are unordered – items can’t be indexed as there’s no structure | *# prints {65, 50, 35, 85}*  grades\_set = {35, 50, 65, 85, 85, 85}  print(grades\_set) | Week 4 |
| Dictionary  using curly brackets {}, but store data values in key:value pairs | gradebook\_dict = {  'Jing': {  'address': 'Fake Rd',  'student\_id': 12345,  'ass1\_grade': 20,  'ass2\_grade': 15,  'ass3\_grade': 0,  'final\_grade': 35  },  'Jim': 50,  'Kerrie': 65,  'Rakesh': 85  }  print(gradebook\_dict['Jing']) *# prints {'address': 'Fake Rd', 'student\_id': 12345, 'ass1\_grade': 20, 'ass2\_grade': 15, 'ass3\_grade': 0, 'final\_grade': 35}* | Week 4 |
| User Defined Functions | def function\_name(first\_parameter):  pass | Week 5 |
| Functions   * Arguments | def print\_hello(name):  print('Welcome ' + name + ' to print\_hello function')  print\_hello('Mark') # OR print\_hello(name = 'Mark')  *# prints Welcome Mark to print\_hello function* | Week 5 |
| Functions   * Multiple parameters and multiple arguments | def print\_hello(name, surname):  print('Welcome ' + name + ' ' + surname + ' to print\_hello function')  *# prints Welcome Mark Smith to print\_hello function*  print\_hello('Mark', 'Smith')  *# OR print\_hello(name = 'Mark', surname = 'Smith')*  *# OR print\_hello(surname = 'Smith', name = 'Mark')* | Week 5 |
| Functions   * Returns * Multiple Returns | def find\_max(numbers):  max\_value = numbers[0] *# Initialise the max value to the first number in the list*  for num in numbers:  if num > max\_value:  max\_value = num *# Update the max value if the current number is greater*  return max\_value  numbers = [3, 6, 2, 8, 4, 10, 1  max\_num = find\_max(numbers)  print(max\_num) *# Output: 10*  def exchange\_usd\_to\_aud(usd\_amount, exchange\_rate):  return usd\_amount \* exchange\_rate  aud\_amount = exchange\_usd\_to\_aud(100, 1.433)  print('100 US dollars would give: ' + str(aud\_amount) + ' AUD dollars')  *# prints 100 US dollars would give: 143.3 AUD dollars*  weather\_data = [37, 23, 30]  def weather\_report(weather):  first = 'Today: ' + str(weather[0])  second = 'Tomorrow: ' + str(weather[1])  third = 'Two days after: ' + str(weather[2])  return first, second, third  monday, tuesday, wednesday = weather\_report(weather\_data)  *# monday = 'Today: 37'*  *# tuesday = 'Tomorrow: 23'*  *# wednesday = 'Two days after: 30'* | Week 5 |
| Types of Arguments   1. Positional arguments:   arguments that are called by their position in the function definition   1. Keyword arguments:   arguments that are called by their name   1. Default arguments:   arguments that are given default values | def travel\_cost(destination\_km, price\_per\_km):  return destination\_km \* price\_per\_km   1. Positional arguments:   travel\_cost(15, 5)   1. Keyword arguments:   travel\_cost(destination\_km = 15, price\_per\_km = 15)  def travel\_cost(destination\_km, price\_per\_km = 5):  return destination\_km \* price\_per\_km   1. Default arguments:   travel\_cost(15)  savings = 1000  def travel\_cost(destination\_km, price\_per\_km = 5):  print('Your savings is: ' + str(savings))  print('Destination (in km) provided: ' + str(destination\_km))  print('Total cost: ' + str(destination\_km \* price\_per\_km))  print(savings)  travel\_cost(20)  *# 1000*  *# Your savings is: 1000*  *# Destination (in km) provided: 20*  *# Total cost: 100* | Week 5 |
| Built-in Functions | print() | Week 5 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |