Topic 2
Variables, Control Flow and Functions

Learning Outcomes

After completing this topic and the recommended reading, you should be able to:

- Import modules, and use them to compute basic statistics.
- Use logic and iteration.
- Identify and use correct syntax and explain the purpose of built-in variable types int, float and list.

1. Variables, Data Types and Structures in Python

Variables

- ***Variable*** is a named piece of memory whose value can change during the running of the program; ***constant*** is a value which cannot change as the program runs.
    - Python doesn't use constant
- Every variable in Python is an object.
- Variable can be rebind to another object of different data type.
- Most variables are immutable objects, i.e., new value is created and rebind to variables, the old value became "garbage".
- Example:

```
pi = 3.14
radius = 11
area = pi * (radius**2)
pi = "hello"
```

Variable Name

- We use ***variable names*** to represent objects (number, data structures, functions, etc.) in our program, to make our program more readable.
    - All variable names must be one word, spaces are never allowed.
    - Can only contain alpha-numeric characters and underscores.
    - Must start with a letter or the underscores character.
    - Cannot begin with a number.
    - Case-sensitive
    - Standard way for most things named in Python is lower with under
        - Lower case with separate words joined by an underscore
- No use of reserved words.

- if; else; for; def; etc.

Mutability

- ***Mutable***: objects that can be modified.
  - Python list and dictionary are mutable objects.
  - Example:

```
colours = ["red", "blue", "green"]
colours[0] = "orange"
```

- ***Immutable***:
  - Python tuple and others are immutable objects.
  - Example:

```
colours = ("red", "blue", "green")  # created as Python tuple
colours[0] = "orange"               # Error: tuple is immutable
```

Aliasing

- Assign a variable to another variable
- Object type: A reference/pointer is copied.
  - Example:

```
colours_1 = ["red", "blue", "green"]
colours_2 = colours_1
colours_2[0] = "orange"
print(colours_2)
print(colours_1)
```

- "Non-object" type: A duplicate is copied when changes is made.
  - R program example:

```
colours_1 <- c("red", "blue", "green")
colours_2 <- colours_1
colours_2[1] <- "orange"
print(colours_2)
print(colours_1)
```

- Make a copy of the object

- Example:

```
colours_3 = colours_1.copy()
colours_3[0] = "black"
print(colours_3)
print(colours_1)
```

- https://pythontutor.com/live.html#mode=edit

Comments

- Not processed by the computer, valued by other programmers.
- Header comments
  - Appear at beginning of a program or a module
  - Provide general information
- Step comments or in-line comments
  - Appear throughout program
  - Explain the purpose of specific portion of code
- Often comments delineated by
  - // comment goes here
  - /* comment goes here */
  - # Python uses this

Python Operations

- Assignment Operator
  - "="
  - Example:
    - a = 67890/12345

      # compute the ratio, store the result in ram, assign to a

      # the value of a is 5.499392
    - b = a

      # b pointing to value of a

- Output
  - "print()"
  - Example:
    - print('Hello World!') *# print the string literals*
    - print(a) # print the value of a

Data Types in Python

- Declaration of variables in Python is not needed
  - Use an assignment statement to create a variable

- **Float**
  - Stores real numbers
  - *a = 4.6*
  - *print(type(a))*

```
<class 'float'>
```

- **Integer**
  - Stores integers
  - *b = 10*
  - *print(type(b))*

```
<class 'int'>
```

- Conversion
  - *int(a)* *# convert float to int* => 4
  - *float(b)* *# convert int to float* => 10.0

- Basic arithmetic operators
  - *3 + 2* # *Addition* => 5
  - *5 – 2* # *Subtraction* => 3
  - *5 * -2* # *Multiplication* => -10
  - *5 / 2.5* # *Division* => 2.0
  - *2**2* # *Exponentiation* => 4
  - *10 % 3* # *Modulus* => 1
  - *10 // 3* # *Floor Division* => 3

- **String**
  - Stores strings
  - *phrase = 'All models are wrong, but some are useful.'*
  - *phrase[0:3]* # *slicing character 0 up to 2*
    => All
  - *phrase.find('models')* # *find the starting index of word*
    => 4
  - *phrase.find('right')* # *word not found*
    => -1
  - *phrase.lower()* # *set to lower case*
    => 'all models are wrong, but some are useful.'
  - *phrase.upper()* # *set to upper case*
    => 'ALL MODELS ARE WRONG, BUT SOME ARE USEFUL.'
  - *phrase.split(',')* # *split strings into list, base on delimiter*
    => ['All models are wrong', ' but some are useful.']

- **Boolean**
  - Stores logical or Boolean values of *TRUE* or *FALSE*
  - *k = 1 > 3*
  - *print(k)*

```
False
```

  - *print(type(k))*

```
<class 'bool'>
```

- Logical operators
  - Conjunction (AND): "**and**"
  - Disjunction (OR): "**or**"
  - Negation (NOT): "**not**"

a	b	a and b	a or b	not a
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

Data Structures in Python

- **Tuples**
  - Store ordered collection of objects
  - Immutable: elements cannot be modified, added or deleted
  - Written with round brackets "( )"
    - *tuple1 = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")*
    - *tuple2 = ("Handsome Koh", 4896, 13.14, True)*

- Accessing elements by indexing
  - *tuple1[0]* *# first element index* => 'apple'
  - *tuple1[-1]* *# last element index* => 'mango'
  - *tuple1[2:5]* *# range of elements* => ('cherry', 'orange', 'kiwi')

- **Lists**
  - Store ordered collection of objects; mutable
  - Written with square brackets "[ ]"
    - *list1 = ["apple", "banana", "cherry"]*
    - *list2 = ["Handsome Koh", 4896, 13.14, True]*
  - Changing elements
    - *list1.append("orange")* *# add to last position*
      => ['apple', 'banana', 'cherry', 'orange']
    - *list1[2] = "coconut"* *# modify index element*
      => ['apple', 'banana', 'coconut', 'orange']
    - *list1.remove("apple")* *# delete elements*
      => ['banana', 'coconut', 'orange']
    - *list1.insert(2, "durian")* *# insert element at position*
      => ['banana', 'coconut', 'durian', 'orange']

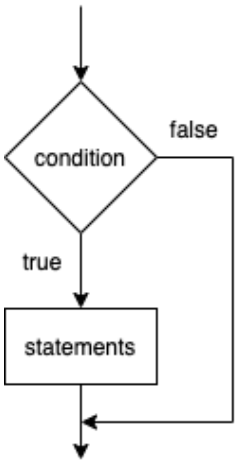
- **Sets**
  - Store unordered, unindexed, nonduplicates collection of objects
  - Written with square brackets "{ }"
    - *set1 = {"apple", "banana", "cherry"}*
    - *set2 = {"apple", "samsung"}*

- Set operations
  - *set1.union(set2)* *# Union both sets*
    => {'apple', 'banana', 'cherry', 'samsung'}
  - *set1.intersection(set2)* *# Intersect both sets*
    => {'apple'}

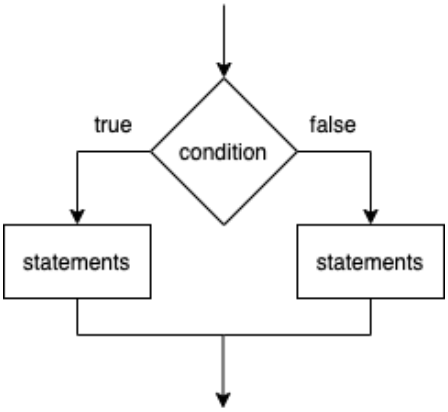
- **Dictionaries**
  - Store unordered collection of objects
  - Written with square brackets "{ }", and "key:value" pair
    - *thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964}*
  - Accessing/modifying elements by key name
    - *thisdict["model"]* => 'Mustang'
    - *thisdist["year"] = 2018* *thisdist["color"] = "red"* => {'brand': 'Ford', 'model': 'Mustang', 'year': 2018, 'color': 'red'}

2. Condition/Decision/Selection/Branching in Python

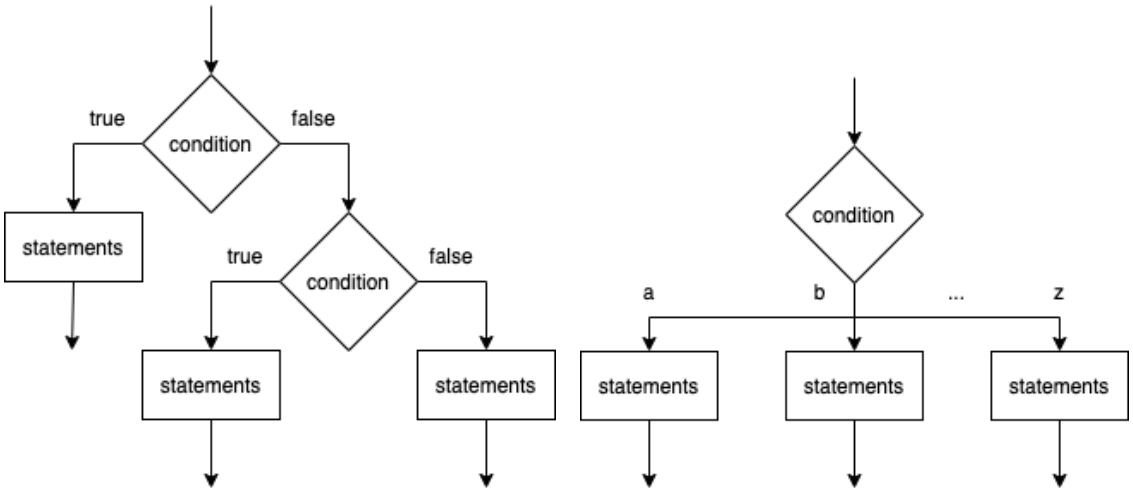
- A ***selection*** (or ***decision***) structure allows a program to perform specific actions only under certain conditions.
    - Evaluate one or more alternatives.
    - Selectively execute statements.
    - Requires a ***condition*** (to test if it is ***true*** or ***false***) to determine when to execute statements, or to decide on the path to be taken.
- Single alternative
    - A single block of statements to be executed or skipped.
    - Example: ***if***



- Dual alternative
    - Two blocks of statements, one of which is to be executed, while the other one is to be skipped.
    - Example: ***if…then…else***



- Multiple alternative
    - More than two blocks of statements, only one of which is to be executed and the rest skipped.
    - Example: ***if…then…else if…else***; ***switch case***



Control Flow Structures in Python

- No parentheses are needed to enclose the control flow structures
- **:** is needed after the **if**, **elif**, **else**, **for**, **while** statements
- Indentation is required for the block after the control flow statement

Conditional Statements

- Single alternative

- Python syntax:

```
if condition:
    true statements
```

- Example:

```
mark = 80
if mark >= 50:
    print("pass")

mark = 30
if mark >= 50:
    print("pass")
    print("congrats!")

mark = 30
if mark >= 50:
    print("pass")
print("congrats!")
```

- Dual alternative
  - Python syntax:

```
if condition:
    true statements
else:
    false statements
```

  - Example:

```
mark = 40
if mark >= 50:
    print("pass")
else:
    print("fail")
```

- Multiple alternative (nested if)
  - Python syntax:

```
if first condition:
    1st condition true statements
elif second condition:
    1st condition false but
    2nd condition true statements
else:
    all false statements
```

- Example:

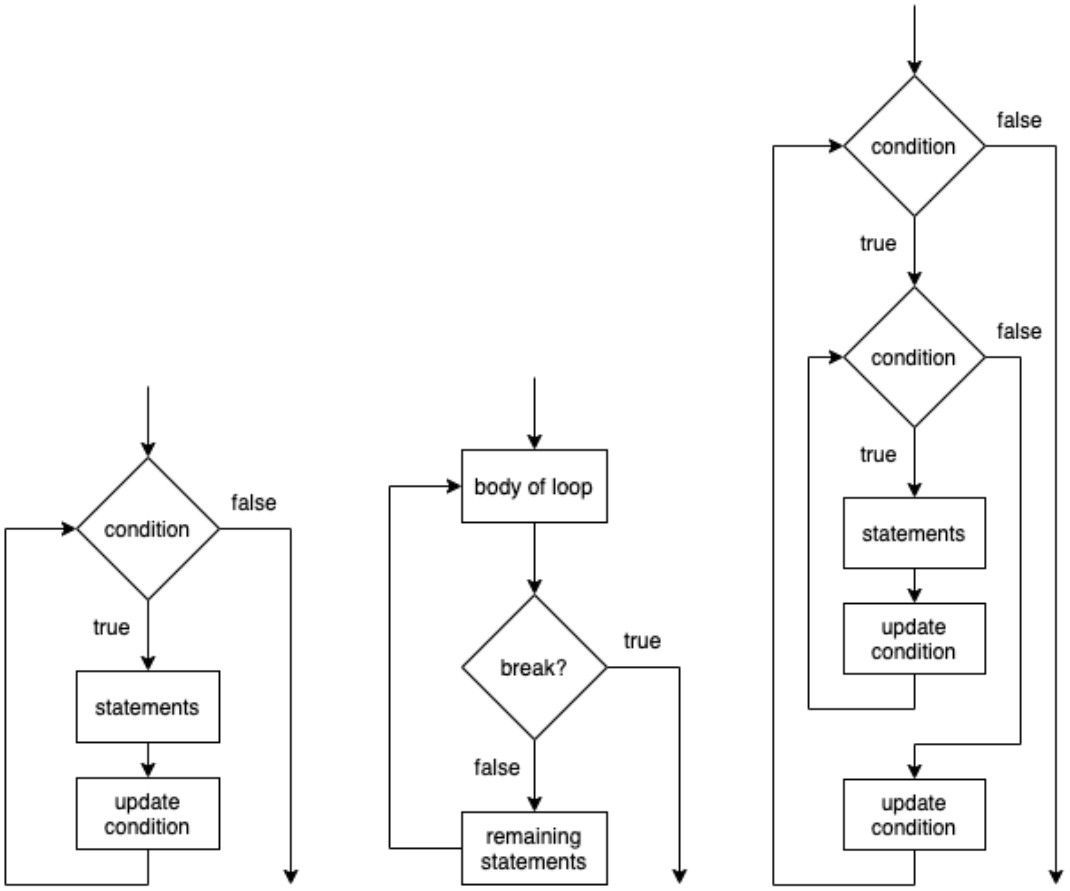
```
mark = 65
if mark >= 70:
    print("distinction")
elif mark >= 60:
    print("merit")
elif mark >= 50:
    print("pass")

mark = 45
if mark >= 70:
    print("distinction")
else:
    if mark >= 60:
        print("merit")
    else:
        if mark >= 50:
            print("pass")
        else:
            print("fail")
```

3. Iteration/Repetition/Loop in Python

- A ***repetition*** (or ***loop***) structure causes a statement or set of statements to execute repeatedly, under some conditions.
    - Repeat statements more than once.
    - Needs a ***stop condition***, i.e., the program will continue to loop until some condition is met.

      → Be careful not to create infinite loops
- Allow a programmer to avoid duplicate code.
    - Duplicate code makes a program large.
    - Write a long sequence of statements is time consuming.
    - If part of the duplicate code must be corrected or change, then the changes must be done many times.
- Condition-controlled loops
    - Test condition before/after performing an iteration.
    - Condition tested for ***true*** or ***false*** value.
    - Example: ***while*** loop; ***repeat*** loop
- Count-controlled loops
    - Iterates a specific number of times, moving through all the elements of data structure.
    - Example: ***for*** loop
- Nested loops
    - All loops can be nested, that is, a loop contained inside of another loop.
    - Inner loop goes through all its iterations for each iteration of outer loop.
    - Inner loops complete their iterations faster than outer loops.
    - Often used to step through two-dimensional arrays.



Iteration Structures in Python

- Count-controlled Loop
  - Python syntax:

```
for item in container:
    statements
```

  - Example:

```
for i in range(1,6):
    print(i)
```

- Terminating early
  - ***continue***: stops the current iteration and moves to next iteration.
  - ***break***: stops the loop entirely.

- Example:

```
for i in [1,2,3,4,5]:
    if i < 2:
        continue
    print(i)
    if i >= 4:
        break
```

- Condition-controlled Loop
  - Python syntax:

```
while condition:
    statements
```

  - Example:

```
word = input("Give me a 4-letter word: ")
while len(word) != 4:
    print("Wrong input!")
    word = input("Give me a 4-letter word: ")
print(word)
```

4. Functions in Python

Writing a user-defined Functions

- Arguments are the inputs to the function module.
- Return statement are the output from the function module.
- Function Definition
    - Python syntax:

```
def function_name(arguments):
    '''
    docstring
    '''
    body of codes

    return value
```

    - Example:

```
def circle_area(radius):
    '''
    This function calculates the area of a circle given the radius
    input: radius
    output: area = pi * radius^2
    '''
    pi = 3.14
    return pi * radius**2

circle_area(2)              # output: 12.56
```

Scope of Function

- Global scope
    - Variables from global scope is available everywhere.
    - Variables created outside of a function can be used by any functions.

- Example:

```
def add_a(x):
    x = x + a       # 'a' refers to global 'a' which has value 8
    return x

a = 8
z = add_a(10)
print(z)
```

- Local scope
  - Variables from local scope is not available elsewhere.
  - Variables created inside of a function cannot be used outside of the function.
  - Example:

```
def add_b(x):
    b = 8
    x = x + b
    return x

z = add_b(10)
print(b)            # Error: 'b' is not defined
```

- LEGB Rule
  - Local → Enclosing → Global → Built-in scope
  - Example:

```
def add_1(x):       # 'x' here has local scope
    x = x + 1
    return x

x = 10              # 'x' here has global scope
z = add_1(12)
print(z)
```

  - https://pythontutor.com/live.html#mode=edit

Functions with Mutable Object

- Pass as reference
  - A pointer to the object is passed to the arguments of function.
  - Changes made in function modified the original object.

- Example:

```
def add_one_number(seq, num):
    seq.append(num)
    return (seq)

nums = [1,3,2,4]
new_nums = add_one_number(nums, 5)
print(new_nums)
print(nums)
```

- Make a copy
  - A new copy of the reference object is made, changes will not modify the original object.
  - Example:

```
def add_one_number(seq, num):
    new_seq = seq.copy()
    new_seq.append(num)
    return (new_seq)

nums = [1,3,2,4]
new_nums = add_one_number(nums, 5)
print(new_nums)
print(nums)
```

  - https://pythontutor.com/live.html#mode=edit

Libraries

- ***SciPy***
  - Python library used for scientific computing
  - Contains modules for optimization, linear algebra, integration, interpolation, etc
- ***NumPy***
  - Python library used for working with arrays
  - Performs numerical processing
- ***Pandas***
  - Python library used for data manipulation and analysis
  - Work with heterogenous data, as data frame

5. Exercises

2.205 Programming Exercise

- Write a piece of code that can compute the highest of two values as a Jupyter Notebook.

2.403 Functions and Libraries

- Refers to “2.403 Topic 2 – lab 1.html”

2.405 Date and Time Data

- Refers to “2.405 Topic 2 – lab 2 datetime.html”

6. Practice Quiz

- Work on *Practice Quiz 02* posted on Canvas.

Useful Resources

-
    - http://