Topic 2

Data Types and Control Structures

Murdoch
UNIVERSITY

LAST WEEK

Murdoch UNIVERSITY

- What is computation
- Algorithms and how to express them
- Stored program computer and basic computer architecture
- Programming language syntax and semantics
- Python programming language
- Compiled language vs interpreted language
- How to run a Python program

THIS WEEK

Murdoch UNIVERSITY

- Data
- Data Types
- Objects
- Variables
- Simple operations
- Expressions
- Assignment statement
- Output statement `print( … )`
- Branching and conditionals
- Indentation
- Iteration and loops

DATA

Murdoch UNIVERSITY

- A computer program gets input data, manipulates the data, and finally produces output data.
- A piece of data can be very simple, such as
  - 12 — an integer
  - 34.5 — a floating point number
  - True — logical true
  - "Python is fun!" — a string
- or it can be complicated, such as
  - ("Apple", "Cherry", "Banana") — a tuple
  - { "John", 25 } — a set
- or very complicated, such as
  - [1, 2, ["ab", "cd", (20, 30, 40)]] — a nested list

DATA TYPES

Murdoch UNIVERSITY

- As you can see, there are different categories of data, eg:
  - integers: 1, 0, -10, -2000003, 98000
  - floating point numbers: 2.3, 501.120, -12.78
  - booleans: True, False
  - strings: "Hong", "Python is fun!"
  - tuples: ("apple", "cherry", "orange")
  - lists: [1, 3, 5, 7, 9], ["abc", "xyz", 3, 4]
  - {"name": "John", "age" : 29, "citizen" : "Asutalian" }

DATA TYPES

Murdoch UNIVERSITY

- In Python, each category is called a data type and each data type is given a name:
  - integers: `int`
  - floating point numbers: `float`
  - booleans: `bool`
  - strings: `str`
  - tuples: `tuple`
  - lists: `list`
  - Dictionary: `dict`

DATA TYPES

Murdoch UNIVERSITY

- Each data type defines two things: a set of data values and a set of operations that can be performed on these data values
- Eg, `bool` type:
  - the set of values: `True`, `False`
  - the set of operations: `and`, `or`, `not`, `==`, `!=`, `etc`
- Eg, `int` type:
  - the set of values: from 0 to any integer number, positive or negative, as long as there are enough memory to store
  - the set of operations: `+`, `-`, `*`, `/`, `%`, `**`, `<`, `<=`, `>`, `>=`, `==`, `!=`, `is`, `is not`, `etc.`

DATA OBJECTS

Murdoch UNIVERSITY

- In Python, every piece of data value is an object.
- programs manipulate data objects
- objects have a type that defines the kinds of things programs can do to them
  - Lecturer is a human, and he/she can walk, speak, give lectures, etc.
  - A chair has four legs and it can be seated by someone.
- objects are either
  - scalar (cannot be subdivided) or
  - non-scalar (have internal structure that can be accessed)

SCALAR OBJECT TYPES

Murdoch UNIVERSITY

- `int` – represent **integers**, eg, `5`
- `float` – represent **real numbers**, eg, `3.27`
- `bool` – represent **Boolean** values `True` and `False`
- `NoneType` – **special** and has one value, `None`

SCALAR OBJECT TYPES

Murdoch UNIVERSITY

- can use the function `type()` to see the type of an object

```
hong$ python3
Python 3.10.2 (v3.10.2:a58ebcc701, Jan 13 2022, 14:50:16) [Clang
13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> type(5)
<class 'int'>
>>> type(3.0)
<class 'float'>
>>>
```

- Note that the Python interpreter displays <class, 'int'>. In fact, every data type in Python is defined as a class. There are built-in classes and user defined classes. We will discuss user-defined classes in Topic 9.

TYPE CONVERSION (CAST)

Murdoch UNIVERSITY

- can convert object of one type to another
- `float(3)` converts integer `3` to float `3.0`
- `int(3.9)` truncates float `3.9` to integer `3`
- `int("351")` convert string "351" too integer 351

PRINT TO CONSOLE

Murdoch UNIVERSITY

- You can type Python statement directly on the Python shell prompt (>>>). When you press Enter key, the statement is interpreted by the Python interpreter and the result is displayed in the next line (which is not proceeded by the shell prompt). Examples

```
>>> 3+2
5
>>> print(3+2)
5
>>>
```

EXPRESSIONS

Murdoch UNIVERSITY

- combine objects and operators to form expressions
- an expression is evaluated to a value, which has a type
- syntax for a simple expression

```
<object> <operator> <object>
```

OPERATORS ON ints and floats

Murdoch UNIVERSITY

- `i+j` -> the **sum**
- `i-j` -> the **difference**
- `i*j` -> the **product**

if both are ints, result is int
if either or both are floats, result is float

- `i/j` -> **division**

result is float

- `i%j` -> the **remainder** when `i` is divided by `j`
- `i**j` -> `i` to the **power** of `j`

SIMPLE OPERATIONS

Murdoch UNIVERSITY

- parentheses are used to tell Python to do these operations first
- operator precedence without parentheses
  - **
  - *
  - /
  - + and – executed left to right, as appear in expression

BINDING VARIABLE AND VALUE

Murdoch UNIVERSITY

- The equal sign = is an assignment of a value to a variable name

variable value

```
Pi = 3.14159

pi_approx = 22/7
```

- The value is stored in computer memory
- an assignment binds the variable name to the value
- retrieve the value associated with a name or variable by invoking the name, eg, by typing `pi`

```
>>> pi=3.14
>>> pi
3.14
```

ABSTRACTING EXPRESSION

Murdoch UNIVERSITY

- why give names to values of expressions?
- to reuse names instead of values
- easier to change code later
- These names are known as variables (as they may contain different values at different time)

```
>>> pi=3.14
>>> pi
3.14
>>> pi = 3.14159
>>> radius=2.2
>>> area = pi*(radius**2)
>>> area
15.205295600000001
>>>
```

PROGRAMMING VS MATH

Murdoch UNIVERSITY

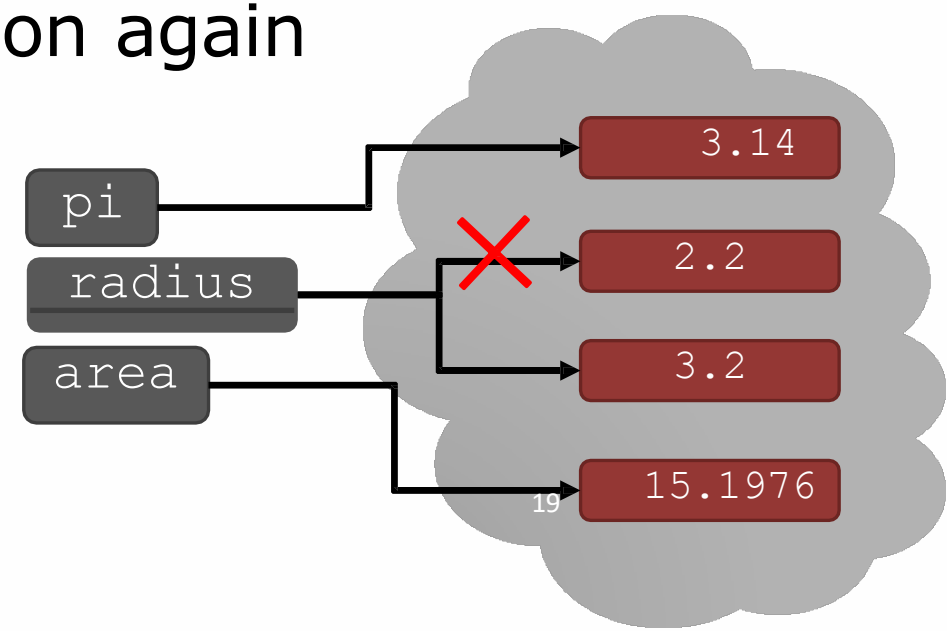
- in programming, you do not “solve for x”
- Unlike in math: `x = 2 - x**2` gives x = 1 or x = -2
- In programming, the expression on the right of an assignment is evaluated to a value, the variable on the left of the assignment represents the memory address of the variable and the value is "assigned" to that address

```
pi = 3.14159
radius = 2.2
area = pi*(radius**2)
radius = radius+1
```

CHANGING BINDING

- can re-bind variable names using new assignment statements
- previous value may still be 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

```
pi = 3.14
radius = 2.2
area = pi*(radius**2)
radius = radius+1
```



STRINGS

Murdoch UNIVERSITY

- A string contains a sequence of letters, special characters, spaces, digits enclosed in single quotes or double quotes

```
hi = 'hello there'
Py = "Python is fun"
```

- **concatenate** two strings

```
name = "buddy"
greet = hi + name
greeting = hi + ' ' + name
```

- do some **operations** on a string as defined in Python docs

```
silly = hi + (" " + name)*3
```

STRINGS

Murdoch UNIVERSITY

- You may use single quotes or double quotes interchangeably. However, if the string contains a double quote character, you may want to use single quotes:

```
hi = '"Good morning", said Mary'
```

- And vice versa:

```
msg = "You've got an error"
```

OUTPUT: print

Murdoch UNIVERSITY

- `print` is a built-in function in Python, it is used to output stuff to the console (eg, terminal)
- We will discuss functions in Topic 4, including how to create your own function

```
x = 5
print(x)
```

- A numeric number such as 5 and a string "5" may look the same when they are printed, but they are very different in terms of how they are stored in computer memory and what types of the operations performed on them.
- Sometimes, we need to convert a number to a string, or a string to a number. Eg, convert a number to a string

```
x_str = str(x)
print("my fav num is " + x_str )
```

INPUT: `input("")`

Murdoch UNIVERSITY

- `input` function prints whatever is in the quotes
- user types in something on the keyboard as input and then hits enter. Then the function returns with a string containing the user input
- 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 (ie, converted to a number)

```
num = int(input("Type a number... "))

print(5*num)
```

Question: what would be printed if no type casting?

ACTIVITY

Murdoch UNIVERSITY

- Write a program that takes the unit code and total mark in that unit as input and prints following customized message.

```
unit = input("Type in your unit code: ")
mark = input("Type in your mark: ")
print("Your unit code is " + unit + "and your mark is " + mark)
```

- Repeat the above, but increase the mark by 10%

COMPARISON OPERATORS ON `int, float, string`

Murdoch UNIVERSITY

- `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`

LOGIC OPERATORS ON bools

Murdoch UNIVERSITY

- `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 only if both a and b are True
a or b -> True if either a or b is True
```

A	B	A and B	A or B
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

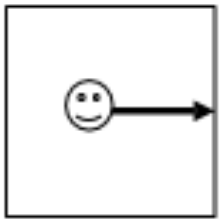
COMPARISON EXAMPLES

Murdoch UNIVERSITY

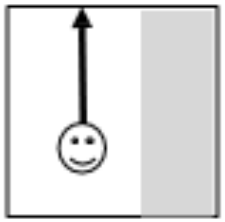
```
pset_time = 15
sleep_time = 8
print(sleep_time > pset_time)


drive = True
drink = False
both = drink and drive
print(both)
```

CONDITION AND BRANCHING



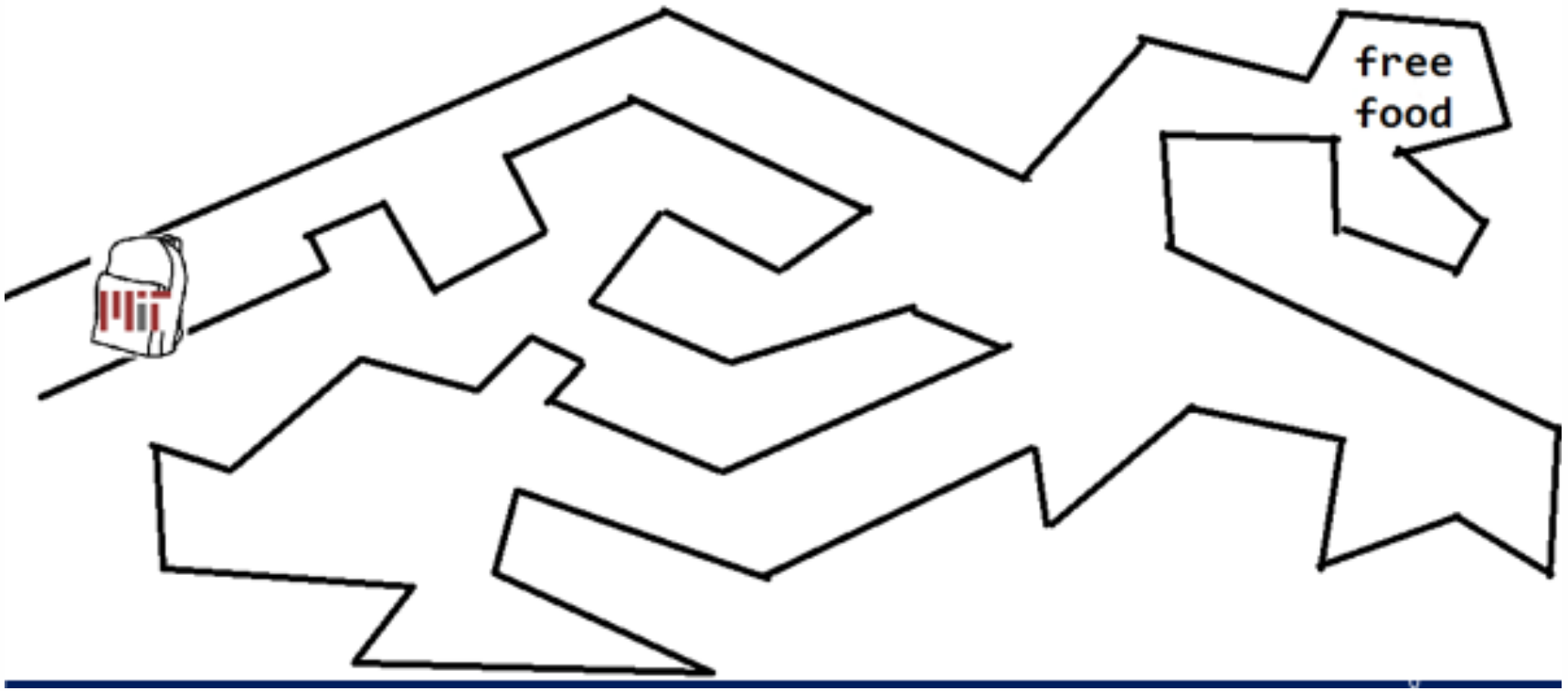
If right clear,
go right



If right blocked,
go forward

If right and
front blocked,
go left

If right , front,
left blocked,
go back



CONTROL FLOW - BRANCHING

Murdoch UNIVERSITY

```
if <condition>:
    <statement>
    <statement>
    ...
```

```
if <condition>:
    <statement>
    <statement>
    ...
else:
    <statement>
    <statement>
```

```
if <condition>:
    <statement>
    <statement>
    ...
elif <condition>:
    <statement>
    <statement>
    ...
else:
    <statement>
    <statement>
    ...
```

- `<condition>` is evaluated to the value `True` or `False`
- Execute the statements in that block if `<condition>` is `True`

INDENTATION AND BLOCK

- matters in Python!!!
- how you denote blocks of code

```
x = float(input("Enter a number for x: "))
y = float(input("Enter a number for y: "))
if x == y:
    print("x and y are equal")
    if y != 0:
        print("therefore, x/y is", x/y)
elif x < y:
    print("x is smaller")
else:
    print("y is smaller")
print("thanks!")
```

What if x = y here?

get a SyntaxError

INDENTATION AND BLOCK

Murdoch UNIVERSITY

- Indentation is important
  - Python replies on the indentation to define the scope of a block
  - Statements in the same block MUST have the same indentation
  - Although the number of spaces in an indentation can vary –
    - it is recommended that you use the same indentation, such as 4 spaces, throughout your program – don't try to be smart!
    - It is also recommended that you use space character rather than tab character to create an indentation.
    - You can define the size of indentation in Idle (go to Preferences)
- Many other programming languages use explicit begin and end symbols to delineate a block: begin/end, {/}, etc.

INDENTATION AND BLOCK

Murdoch UNIVERSITY

- Check the following code and identify errors:

```
x = float(input("Enter a number for x: "))
y = float(input("Enter a number for y: "))
if x == y:
    print("x and y are equal")
     if y != 0:
        print("therefore, x/y is", x/y)
elif x < y:
print("x is smaller")
else:
      print("y is smaller")
 print("thanks!")
```

ACTIVITIES

Murdoch UNIVERSITY

- Write a program that takes the unit code and total mark in that unit as input and prints the letter grade according to following letter grade conversion table:

HD	80-100
D	70-79
C	60-69
P	50-59
N	<50

ACTIVITIES

Murdoch UNIVERSITY

- Solution

```
unit = input("Type in your unit code: ")
mark = int(input("Type in your mark: "))
grade = 'N'
if mark >= 80:
    grade = 'HD'
if mark >= 70 and mark <80:
    grade = 'D'

# complete the rest of the code
# also try to use a single if-elif-else statement to do the same

print("Your unit mark is ", mark, " your unit grade is " + grade)
```

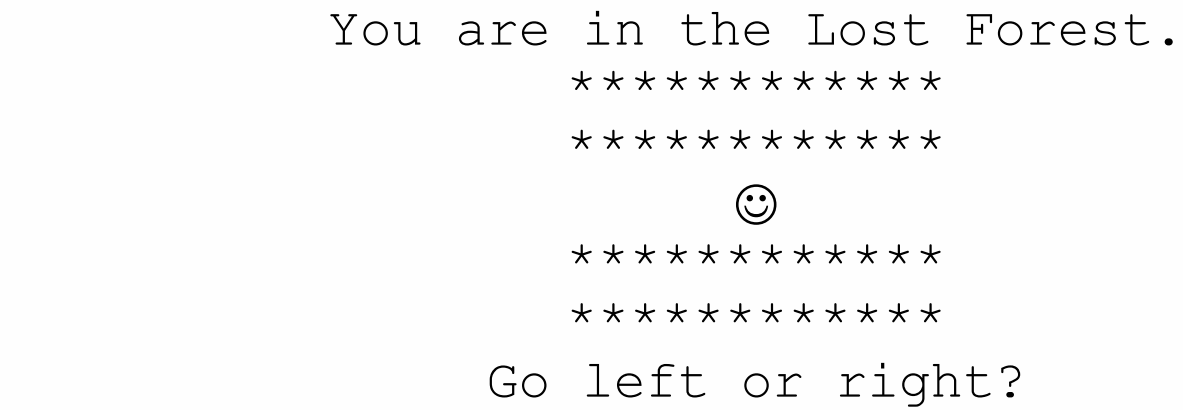
CONTROL FLOW: `while` LOOPS

Murdoch UNIVERSITY

```
while <condition>:
    <statement>
    <statement>
    ...
```

- `<condition>` evaluates to a Boolean
- if `<condition>` is `True`, do all the steps inside the while code block
- check `<condition>` again
- repeat until `<condition>` is `False`

CONTROL FLOW: `while` LOOPS



PROGRAM:

```
n = input("You're in the Lost Forest. Go left or right? ")
while n == "right":
    n = input("You're in the Lost Forest. Go left or right? ")
print("You got out of the Lost Forest!")
```

CONTROL FLOW: `while` LOOPS

Murdoch UNIVERSITY

- iterate through numbers in a sequence

```
# more complicated with while loop
n = 0
while n < 5:
    print(n)
    n = n+1

# shortcut with for loop
for n in range(5):
    print(n)
```

CONTROL FLOW: `for` LOOPS

Murdoch UNIVERSITY

```
for <variable> in range(<some_num>):
    <statement>
    <statement>
    ...
```

- Python function `range` returns a sequence of numbers starting from 0 and incremented by 1 by default
  - So `range(5)` returns the sequence `0,1,2,3,4`
- each time through the loop, <variable> takes a value of the sequence
  - first time, <variable> starts at the 1st value (0)
  - next time, <variable> gets the next value
  - etc.

range(start,stop,step)

Murdoch UNIVERSITY

- default values are `start = 0` and `step` is optional (default to 1)
- loop until value is `stop - 1`

```
mysum = 0
for i in range(7, 10):
    mysum += i # same as mysum = mysum + 1
print(mysum)

mysum = 0
for i in range(5, 11, 2):
    mysum += i
print(mysum)
```

`break` STATEMENT

Murdoch UNIVERSITY

- immediately exits whatever loop it is in
- skips remaining expressions in code block
- exits only the innermost loop!

```
while <condition_1>:
    <statement_b>
    if <condition_2>:
        break
    <statement_c>
print("after while loop")
```

`break` STATEMENT

Murdoch UNIVERSITY

```
mysum = 0
for i in range(5, 12, 2):
    if mysum > 10:
        break
    mysum += i
print(mysum)
```

- what value would be printed in this program?

`for` VS `while` LOOPS

Murdoch UNIVERSITY

`for` loops

- know number of iterations
- can end early via `break`
- uses a counter
- can rewrite a `for` loop using a while loop

`while` loops

- unbounded number of iterations
- can end early via break
- can use a counter but must initialize before loop and increment it inside loop
- may not be able to easily rewrite a while loop using a for loop

ACKNOWLEDGEMENT

Murdoch UNIVERSITY

- Sources used in this presentation include:
- Programiz.com
- MIT OCW