INT3075 Programming and Problem Solving for Mathematics

Control (Part I):
Selection

Control, Quick Overview

Selection

Selection

- Selection is how programs make choices, and it is the process of making choices that provides a lot of the power of computing

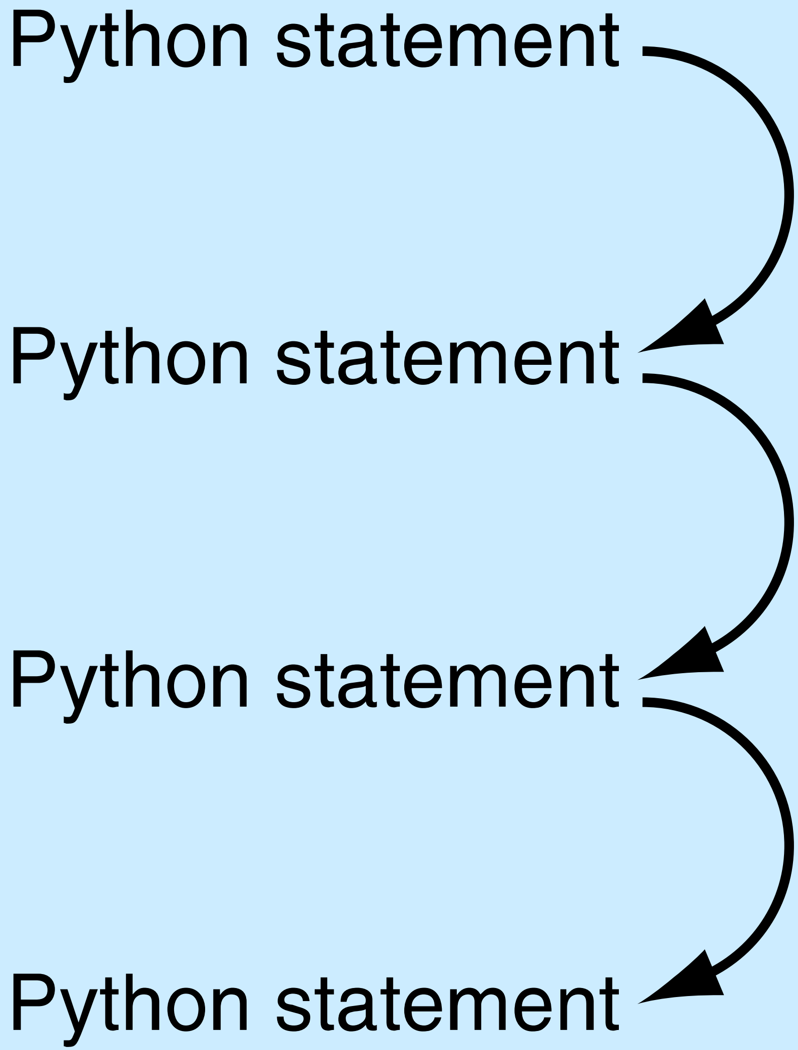


FIGURE 2.1 Sequential program flow.

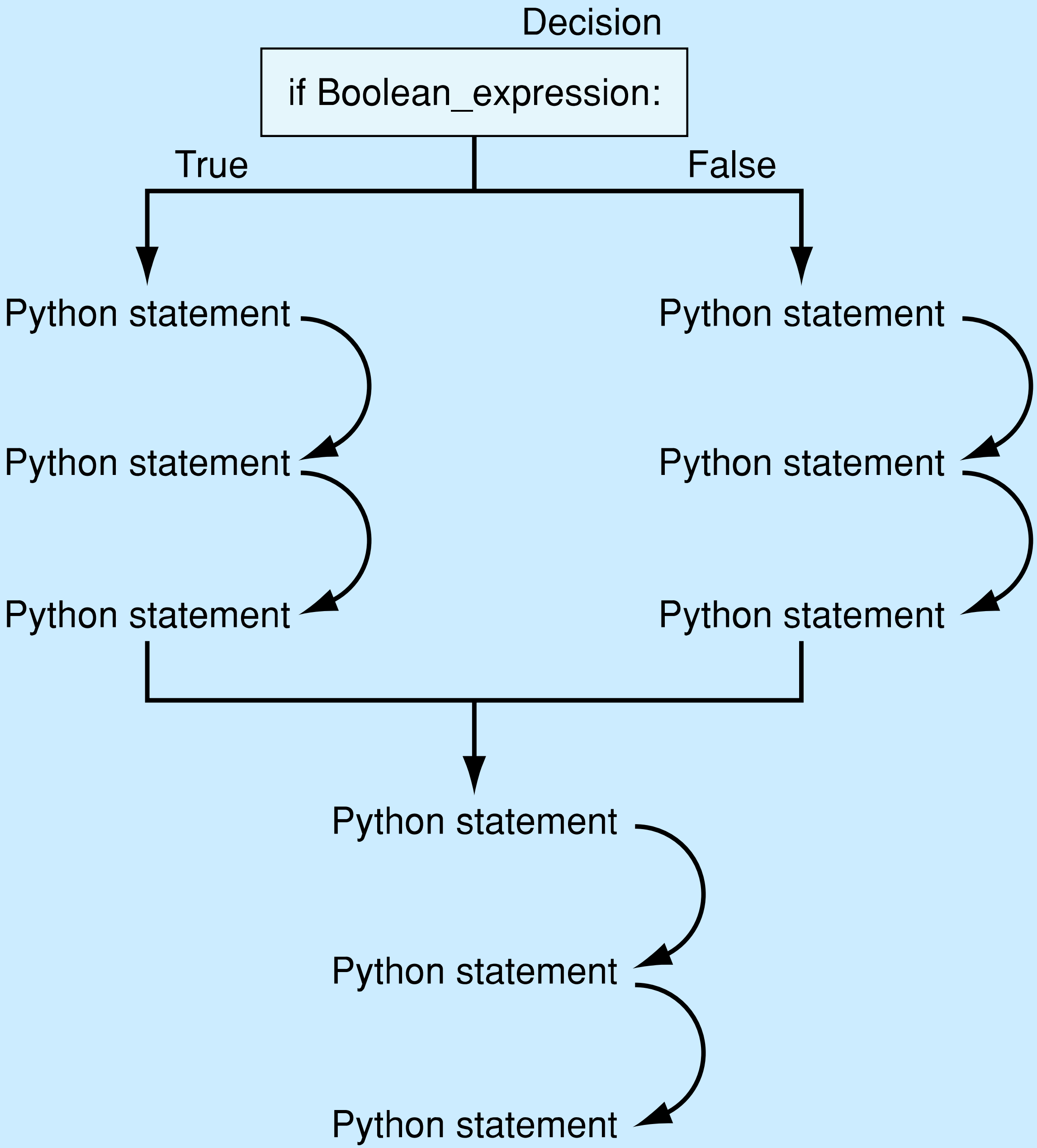


FIGURE 2.2 Decision making flow of control.

<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
==	equal to
!=	not equal to

TABLE 2.1 Boolean Operators.

Note that == is equality,
= is assignment

Python if statement

```
if boolean expression :
    suite
```

- evaluate the boolean (`True` or `False`)
- if `True`, execute all statements in the suite

Warning about indentation

- Elements of the suite must all be indented the same number of spaces/tabs
- Python only recognizes suites when they are indented the same distance (***standard is 4 spaces***)
- You must be careful to get the indentation right to get suites right.

Python Selection, Round 2

```
if boolean expression:
    suite1
else:
    suite2
```

The process is:

- evaluate the boolean
- if `True`, run suite1
- if `False`, run suite2

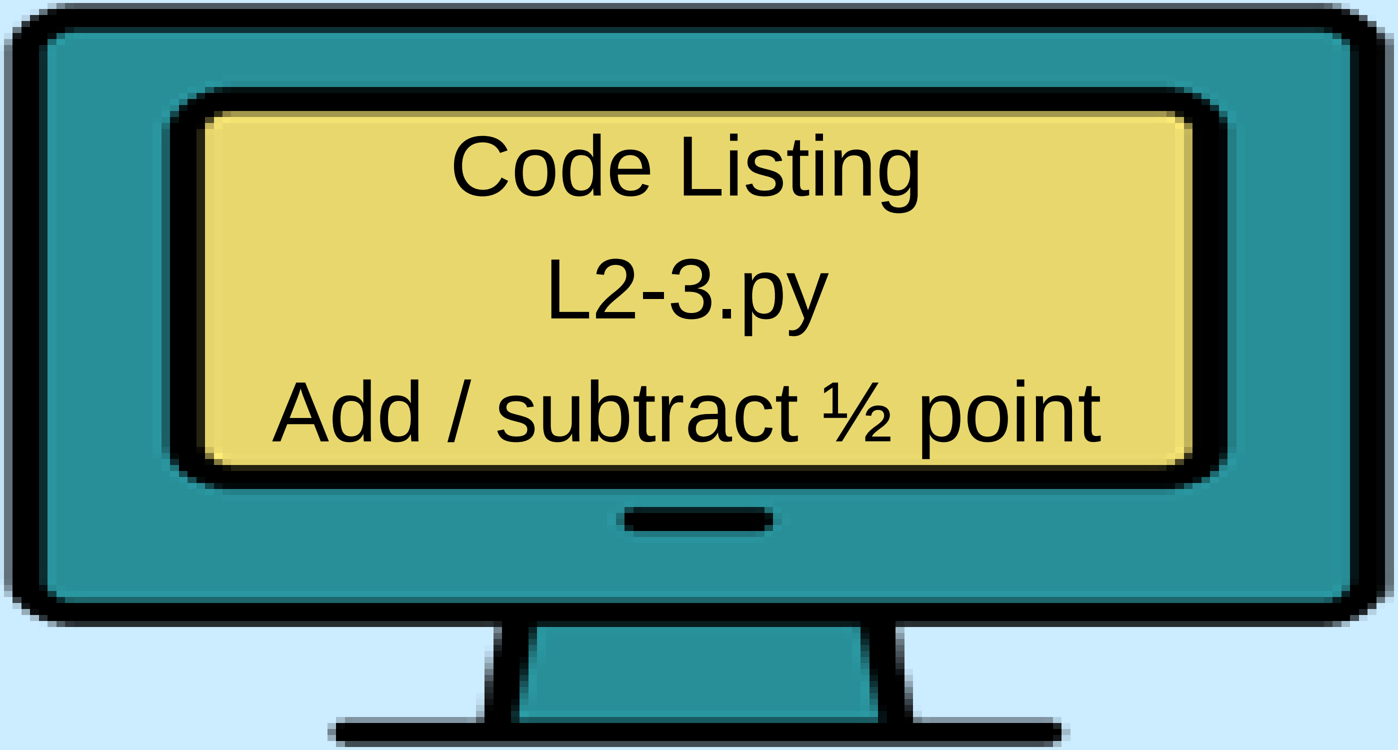
```
>>> first_int = 10
>>> second_int = 20
>>> if first_int > second_int:
        print("The first int is bigger!")
    else:
        print("The second int is bigger!")


The second int is bigger!
>>>
```

Safe Lead in Basketball

- Algorithm due to Bill James (http://www.slate.com/id/2185975/)
- under what conditions can you safely determine that a lead in a basketball game is insurmountable?

The algorithm

- Take the number of points one team is ahead
- Subtract three
- Add ½ point if team that is ahead has the ball, subtract ½ point otherwise
- Square the result
- If the result is greater than the number of seconds left, the lead is safe


Code Listing
L2-3.py
Add / subtract ½ point

first cut

```
# 3. Add a half-point if the team that is ahead has the ball,
#    and subtract a half-point if the other team has the ball.

has_ball_str = input("Does the lead team have the ball (Yes or No): ")

if has_ball_str == "Yes":
    lead_calculation_float = lead_calculation_float + 0.5
else:
    lead_calculation_float = lead_calculation_float - 0.5
```

Problem, what if the lead_calculation_float is less than 0?

Code Listing

L2-4.py

Catch the lead < 0

second cut

```
# 3. Add a half-point if the team that is ahead has the ball,
#    and subtract a half-point if the other team has the ball.

has_ball_str = input("Does the lead team have the ball (Yes or No): ")

if has_ball_str == 'Yes':
    lead_calculation_float = lead_calculation_float + 0.5
else:
    lead_calculation_float = lead_calculation_float - 0.5

# (Numbers less than zero become zero)
if lead_calculation_float < 0:
    lead_calculation_float = 0
```

catch the lead less than 0


Code Listing
L2-7.py
Safe lead program

```python
# 1. Take the number of points one team is ahead.
points_str = input("Enter the lead in points: ")
points_remaining_int = int(points_str)

# 2. Subtract three.
lead_calculation_float= float(points_remaining_int - 3)

# 3. Add a half-point if the team that is ahead has the ball,
#    and subtract a half-point if the other team has the ball.
has_ball_str = input("Does the lead team have the ball (Yes or No): ")

if has_ball_str == 'Yes':
    lead_calculation_float= lead_calculation_float + 0.5
else:
    lead_calculation_float= lead_calculation_float - 0.5

# (Numbers less than zero become zero)
if lead_calculation_float< 0:
    lead_calculation_float= 0

# 4. Square that.
lead_calculation_float= lead_calculation_float** 2

# 5. If the result is greater than the number of seconds left in the game,
#    the lead is safe.
seconds_remaining_int = int(input("Enter the number of seconds remaining: "))

if lead_calculation_float> seconds_remaining_int:
    print("Lead is safe.")
else:
    print("Lead is not safe.")
```

Control in Depth

Booleans

Boolean Expressions

- George Boole's (mid-1800's) mathematics of logical expressions
- Boolean expressions (conditions) have a value of **True** or **False**
- Conditions are the basis of choices in a computer, and, hence, are the basis of the appearance of intelligence in them.

What is True, and what is False

- true: any nonzero number or nonempty object. `1, 100, "hello", [a,b]`
- false: a zero number or empty object. `0, "",[ ]`
- Special values called `True` and `False`, which are just substitutes for 1 and 0. However, they print nicely (`True` or `False`)

Boolean expression

- Every boolean expression has the form:
  - expression booleanOperator expression
- The result of evaluating something like the above is also just true or false.
- However, remember what constitutes true or false in Python!

Relational Operators

- `3 > 2` ➔ `True`
- `8 < 1` ➔ `False`
- `'1' < 2` ➔ Error
    - can only compare like types
- `int('1') < 2` ➔ `True`
    - like types, regular comparison

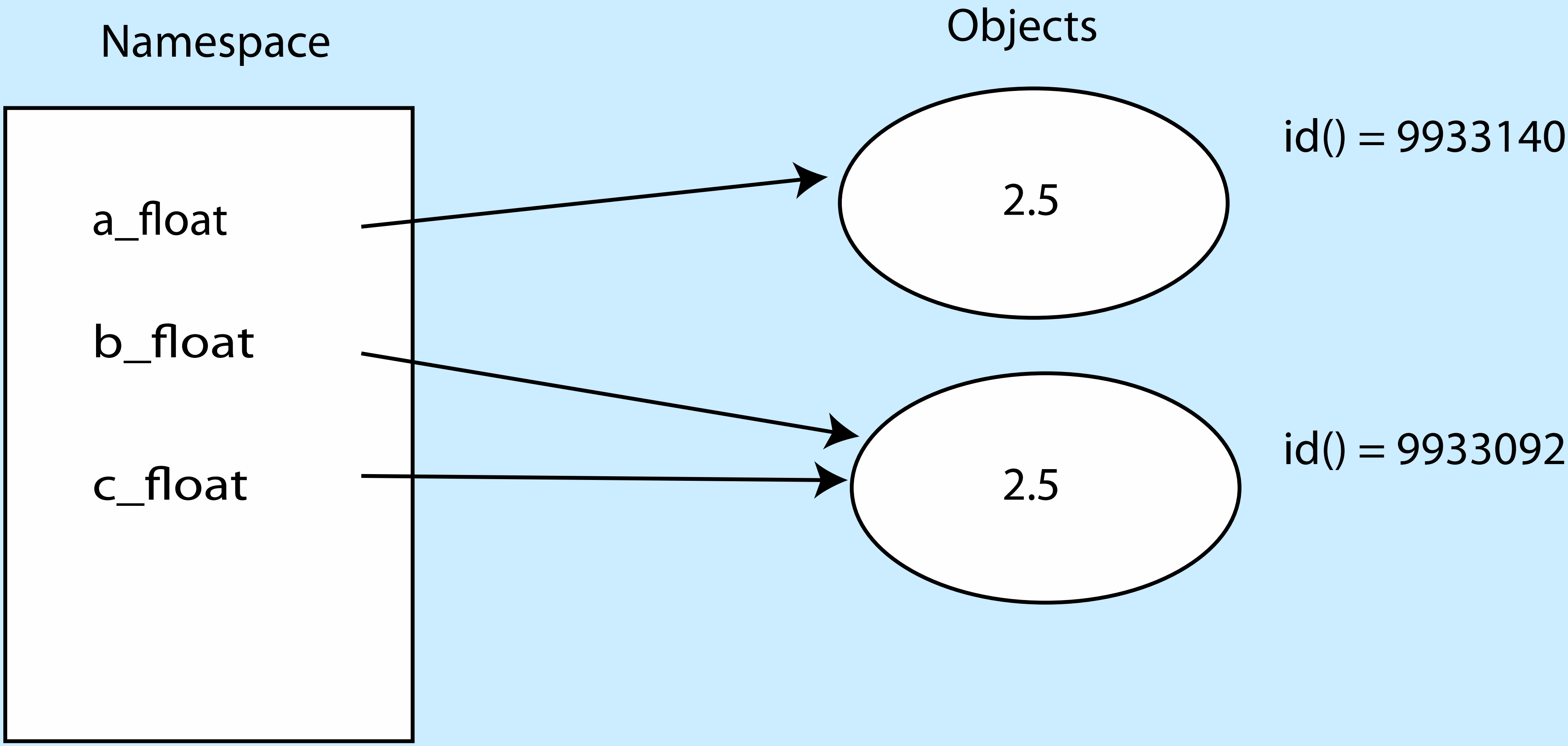
What does Equality mean?

Two senses of equality

- two variables refer to different objects, each object representing the same value
- two variables refer to the same object. The `id()` function used for this.

FIGURE 2.6 What is equality?

```
a_float = 2.5
b_float = 2.5
c_float = b_float
```



equal vs. same

- **==** compares values of two variable's objects to check whether they represent the same value
- **is** operator determines if two variables are associated with the same object

From the figure:

```
a_float == b_float → True
a_float is b_float → False
b_float is c_float → True
```

Pitfall

floating point arithmetic is approximate!

```
>>> u = 11111113
>>> v = -11111111
>>> w = 7.51111111
>>> (u + v) + w
9.51111111
>>> u + (v + w)
9.5111111110448837
>>> (u + v) + w == u + (v + w)
False
```

compare using "close enough"

Establish a level of "close enough" for equality

```
>>> u = 11111113
>>> v = -11111111
>>> w = 7.51111111
>>> x = (u + v) + w
>>> y = u + (v + w)
>>> x == y
False
>>> abs(x - y) < 0.0000001  # abs is absolute value
True
```

Chained comparisons

- In Python, chained comparisons work just like you would expect in a mathematical expression:
- Given myInt has the value 5

```
0 <= myInt <= 5 → True
0 < myInt <= 5 < 1 → False
```

Compound Expressions

Python allows bracketing of a value between two Booleans, as in math

```
a_int = 5
0 <= a_int <= 10   → True
```

- `a_int >= 0` and `a_int <= 10`
- `and`, `or`, `not` are the three Boolean operators in Python

Truth Tables

p	q	not p	p and q	p or q
True	True	False	True	True
True	False	False	False	True
False	True	True	False	True
False	False	True	False	False

Compound Evaluation

- Logically `0 < a_int < 3` is actually
  `(0 < a_int) and (a_int < 3)`
- Evaluate using `a_int` with a value of 5:
  `(0< a_int) and (a_int < 3)`
- Parenthesis first: `(True) and (False)`
- Final value: `False`

Precedence & Associativity

Relational operators have precedence and associativity just like numerical operators.

Operator	*Description*
()	Parenthesis (grouping)
**	Exponentiation
+x, -x	Positive, Negative
*,/,%	Multiplication, Division, Remainder
+,-	Addition, Subtraction
<, <=, >, >=,! =, ==	Comparisons
not x	Boolean NOT
and	Boolean AND
or	Boolean OR

TABLE 2.2 Precedence of Relational and Arithmetic Operators: Highest to Lowest

Boolean operators vs. relationals

- Relational operations always return `True` or `False`
- Boolean operators (`and, or`) are different in that:
  - They can return values (that represent `True` or `False`)
  - They have ***short circuiting***

Remember!

- `0, '',[ ]` or other “empty” objects are equivalent to `False`
- anything else is equivalent to `True`

More on Assignments

Remember Assignments?

- Format: `lhs = rhs`
- Behavior:
  - expression in the rhs is evaluated producing a value
  - the value produced is placed in the location indicated on the lhs

Can do multiple assignments

```
a_int, b_int = 2, 3
```

first on right assigned to first on left, second on right assigned to second on left

`print(a_int, b_int)   #` prints 2 3

`a_int,b_int = 1,2,3` ➔ Error

counts on lhs and rhs must match

traditional swap

- Initial values: `a_int= 2, b_int = 3`
- Behavior: swap values of `X` and `Y`
  - Note: `a_int = b_int`

    `b_int = a_int` doesn't work (why?)
  - introduce extra variable `temp`
    - `temp = a_int`   # save a_int value in temp
    - `a_int = b_int`   # assign a_int value to b_int
    - `b_int = temp`   # assign temp value to b_int

Swap using multiple assignment

```
a_int, b_int = 2, 3
print(a_int, b_int) # prints 2  3

a_int, b_int = b_int, a_int
print(a_int, b_int) # prints 3  2
```

remember, evaluate all the values on the rhs first, then assign to variables on the lhs

Chaining for assignment

Unlike other operations which chain left to right, assignment chains right to left

```
a_int = b_int = 5
print(a_int, b_int)  # prints 5  5
```

More Control: Selection

Compound Statements

- Compound statements involve a set of statements being used as a group
- Most compound statements have:
  - a header, ending with a `:` (colon)
  - a suite of statements to be executed
- `if, for, while` are examples of compound statements

General format, suites

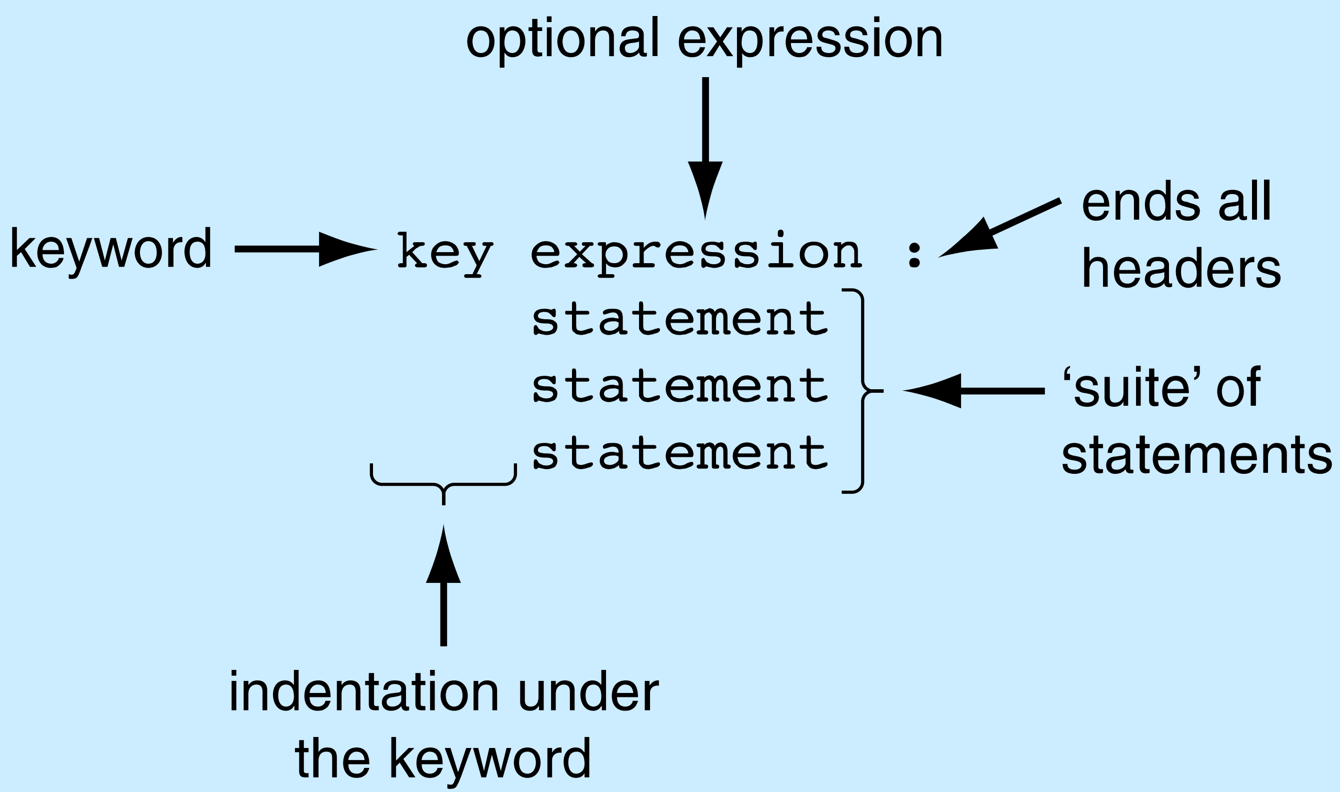


Figure 2.3: Control expression.

Have seen 2 forms of selection

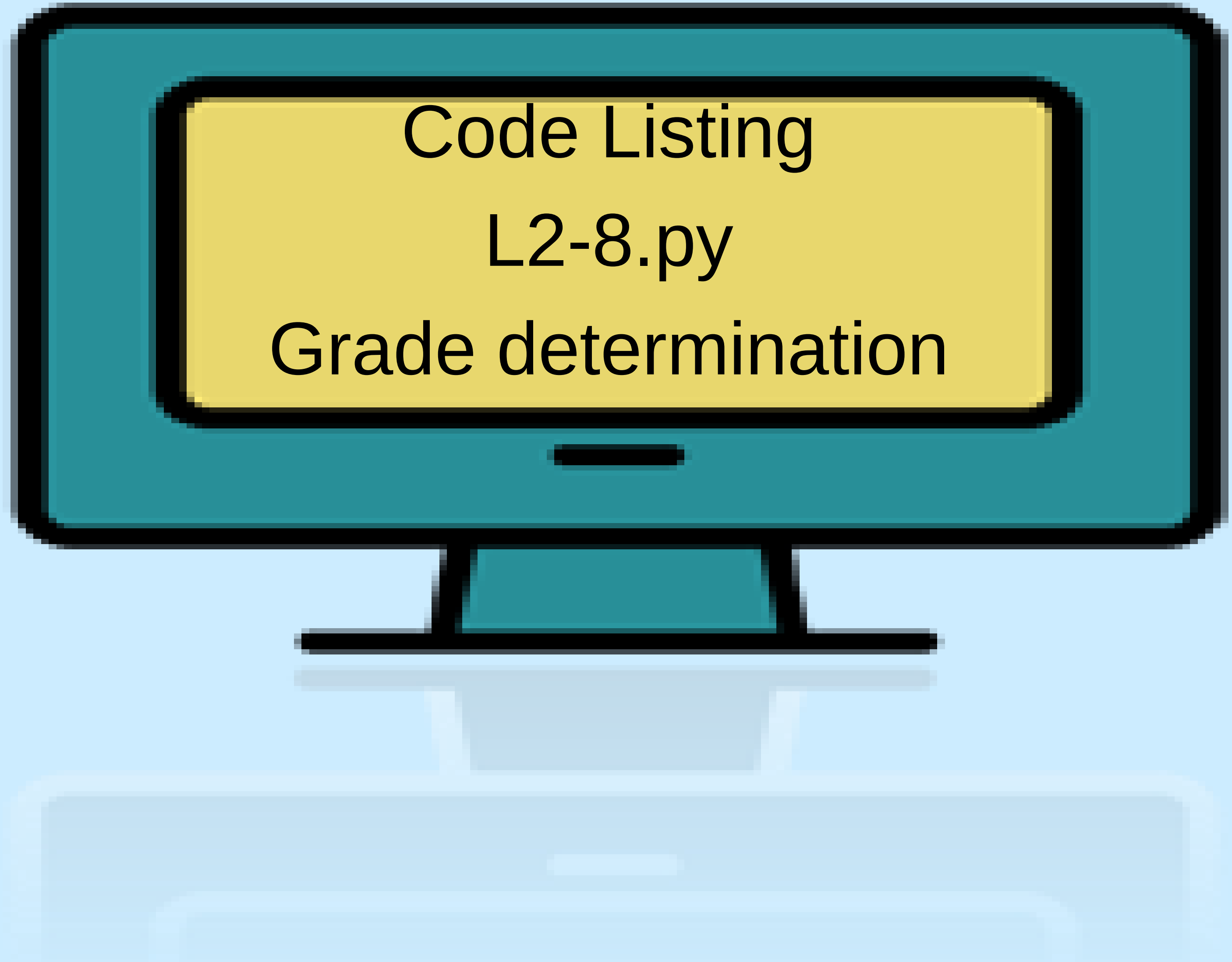
```
if boolean expression:
    suite

if boolean expression:
    suite
else:
    suite
```

Python Selection, Round 3

```
if boolean expression1:
      suite1
elif boolean expression2:
      suite2
(as many elif's as you want)
else:
      suite_last
```

if, elif, else, the process

- evaluate Boolean expressions until:
  - the Boolean expression returns `True`
  - none of the Boolean expressions return `True`
- if a boolean returns `True`, run the corresponding suite. Skip the rest of the `if`
- if no boolean returns `True`, run the `else` suite, the default suite


Code Listing
L2-8.py
Grade determination

```python
percent_float = float(input("What is your percentage? "))

if 90 <= percent_float < 100:
    print("you received an A")
elif 80 <= percent_float < 90:
    print("you received a B")
elif 70 <= percent_float < 80:
    print("you received a C")
elif 60 <= percent_float < 70:
    print("you received a D")
else:
    print("oops, not good")
```

What happens if `elif` are replaced by `if`?

Perfect Number Example

a perfect number

- numbers and their factors were mysterious to the Greeks and early mathematicians
- They were curious about the properties of numbers as they held some significance
- A perfect number is a number whose sum of factors (excluding the number) equals the number
- First perfect number is: 6 (1+2+3)

abundant, deficient

- abundant numbers summed to more than the number.

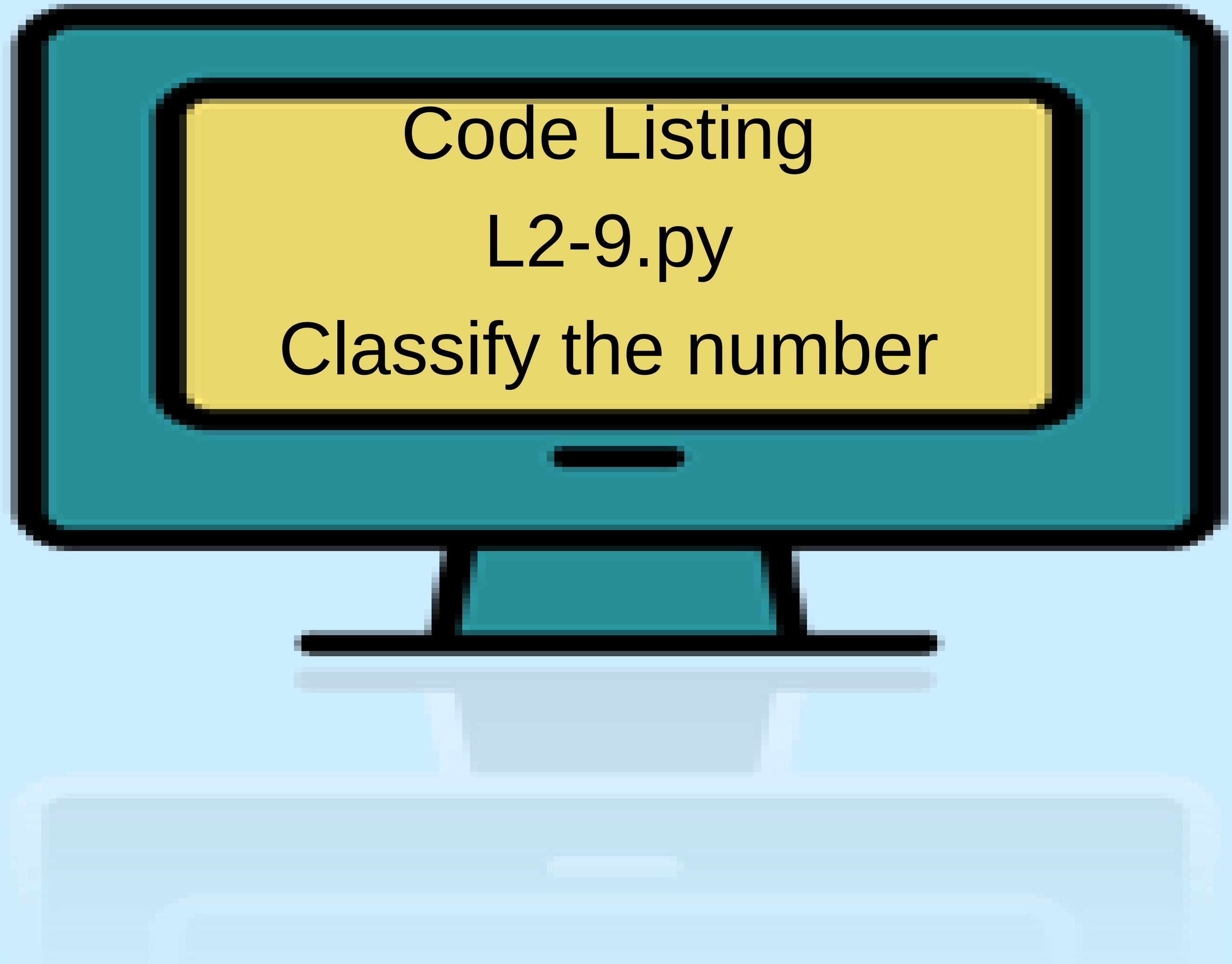
  12: 1+2+3+4+6 =16

- deficient numbers summed to less than the number.

  13: 1

design

- prompt for a number
- for the number, collect all the factors
- once collected, sum up the factors
- compare the sum and the number and respond accordingly


Code Listing
L2-9.py
Classify the number

```
# Classify the number based on its divisor sum
if number_int == sum_of_divisors_int:
    print (number_int, “is perfect”)
else:
    print (number_int, “is not perfect”)
```