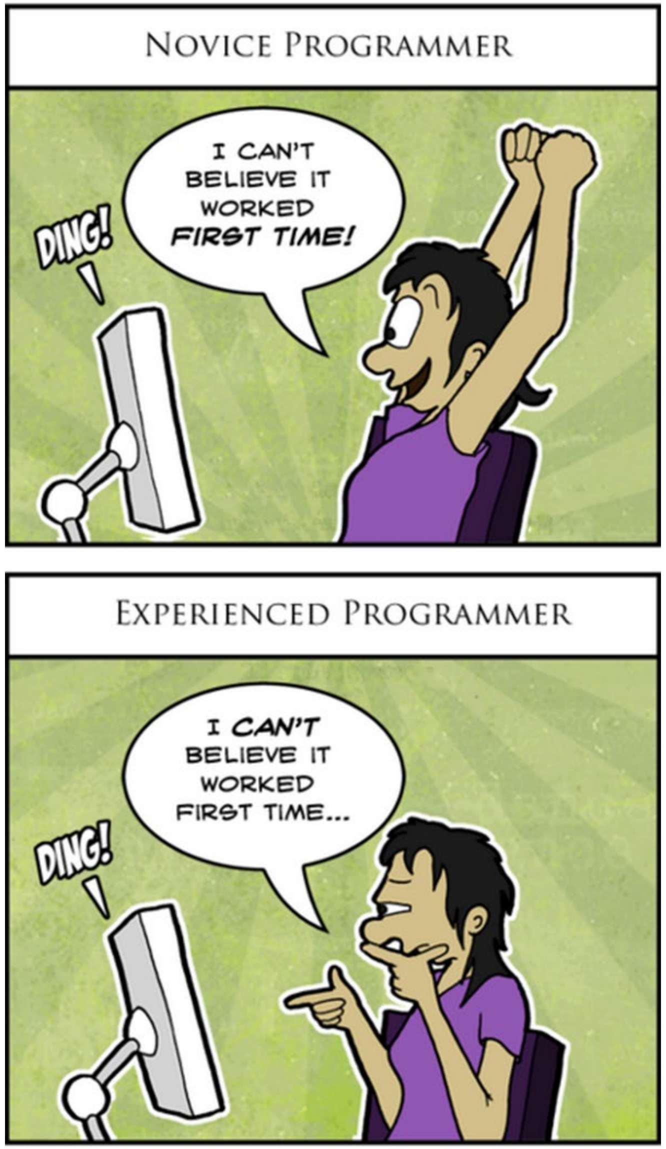
Building Python Programs

Chapter 4: Conditional Execution

Opening Business

- Test next week!
  - Review first day, test second day
- Virtual or In-person Office Hours:
  - Mon-Thur 8:30-10am
- Meetings with TAs:
  - See Discord!


NOVICE PROGRAMMER
DING!
I CAN'T BELIEVE IT WORKED FIRST TIME!
EXPERIENCED PROGRAMMER
DING!
I CAN'T BELIEVE IT WORKED FIRST TIME...

Interactive Programs

Interactive programs

interactive program: Reads input from the console.

- While the program runs, it asks the user to type input.
- The input typed by the user is stored in variables in the code.

- Can be tricky; users are unpredictable and misbehave.
- But interactive programs have more interesting behavior.

`input`

- **`input`**: A function that can read input from the user.
- **`input`** always returns a string

- Using an `input` object to read console input:

  name `=` `input(`**prompt**`)`

  - Example:

    ```
    name = input("type your name: ")
    ```

    - The variable `name` will store the value the user typed in

input example

```
def main():
    age = input("How old are you? ")

    years = 65 - age
    print(years, " years until retirement!")
```

age 29

- Console (user input underlined):

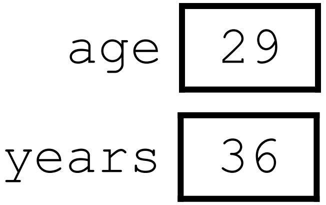
```
How old are you? 29
```

```
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    print(65 - age)
TypeError: unsupported operand type(s) for -:
'int' and 'str'
```

input example

```
def main():
    age = int(input("How old are you? "))

    years = 65 - age
    print(years, "years until retirement!")
```



age 29

years 36

- Console (user input underlined):

```
How old are you? 29
36 years until retirement!
```

Random

Pseudo-Randomness

- Computers generate numbers in a predictable way using a mathematical formula

- Parameters may include current time, mouse position
    - In practice, hard to predict or replicate

- True randomness uses natural processes
    - Atmospheric noise (http://www.random.org/)
    - Lava lamps (patent #5732138)
    - Radioactive decay

Random

- `random` generates pseudo-random numbers.
  - `random` can be accessed by including the following statement:
    `import random`

Method name	**Description**
`random.random()`	returns a random float in the range [0, *1*) in other words, 0 inclusive to *1* exclusive
`random.randint(`***min, max***`)`	returns a random integer in the range [min, *max*] in other words, min to *max* inclusive

  - Example:

```
import random
random_number = random.randint(1, 10)   # 1-9
```

Generating random numbers

- To get a number in arbitrary range [*min*, *max*] inclusive:

  `random.randint(`***min, max***`)`

  - Where ***size of range*** is (***max*** `-` ***min*** `+ 1`)

- Example: A random integer between 4 and 10 inclusive:

```
n = random.randint(4, 10)
```

Exercise

- Write a program that prompts the user for a width and height and then outputs the area of a box with the specified dimensions. Example execution see below:

```
Width? 3

Height? 4

The area is 12.
```

Exercise

- Write a program that prompts the user for a character 5 times. It should output the first word once, the second twice, the third three times, the fourth four times and the fifth five times. Example execution see below:

```
Word? the
the
Word? wizard
Wizardwizard
Word? of
Ofofof
Word? Oz
OzOzOzOz
Word? CSc
CScCScCScCScCSc
```

Exercise

- Write a program that prompts the user for height and width of a multiplication table. It should output a box of integers that match the dimensions. Example execution see below:

```
Width? 7
Height? 5

1  2  3  4  5  6  7
2  4  6  8  10 12 14
3  6  9  12 15 18 21
4  8  12 16 20 24 28
5  10 15 20 25 30 35
```

Strings

Strings

- **string**: a type that stores a sequence of text characters.

  name = "text"

  name = expression

  - Examples:

    ```
    name = "Daffy Duck"
    ```

    ```
    x = 3
    y = 5
    point = "(" + str(x) + ", " + str(y) + ")"
    ```

Indexes

- Characters of a string are numbered with 0-based *indexes*:

```
name = "Ultimate"
```

index	0	1	2	3	4	5	6	7
	-8	-7	-6	-5	-4	-3	-2	-1
character	U	l	t	i	m	a	t	e

  - First character's index : 0
  - Last character's index : 1 less than the string's length

Accessing characters

- You can access a character with **string**[**index**]:

```
name = "Merlin"
print(name[0])
```

Output: `M`

Accessing substrings

- Syntax:

```
part = string[start:stop]
```

- Example:

```
s = "Merlin"
mid = s[1:3]      # er
```

- If you want to start at the beginning you can leave off start

```
mid = s[:3]       # Mer
```

- If you want to start at the end you can leave off the stop

```
mid = s[1:]       # erlin
```

`String` methods

Method name	Description
`find(`**`str`**`)`	index where the start of the given string appears in this string (-1 if not found)

`substring(`**`index1`**`, `**`index2`**`)`
or
`substring(`**`index1`**`)`
	the characters in this string from *index1* (inclusive) to *index2* (exclusive);
	if *index2* is omitted, grabs till end of string

`lower()`	a new string with all lowercase letters
`upper()`	a new string with all uppercase letters

- These methods are called using the dot notation below:

```
starz = "Biles & Manuel"
print(starz.lower())    # biles & manuel
```

String method examples

```
# index     012345678901
s1 = "Allison Obourn"
s2 = "Merlin The Cat"

print(s1.find("o"))     # 5
print(s2.lower())       # "merlin the cat"
```

- Given the following string:

```
# index  012345678901234567890123
book =  "Building Python Programs"
```

  - How would you extract the word "Python" ?

Modifying strings

- String operations and functions like `lowercase` build and return a new string, rather than modifying the current string.

```
s = "Aceyalone"
s.upper()
print(s)    # Aceyalone
```

- To modify a variable's value, you must reassign it:

```
s = "Aceyalone"
s = s.upper()
print(s)    # ACEYALONE
```

Other `String` operations - length

- Syntax:

```
length = len(string)
```

- Example:

```
s = "Merlin"
count = len(s)     # 6
```

Looping through a string

- The `for` loop through a string using range:

```
major = "CSc"
for letter in range(0, len(major)):
    print(major[letter])
```

- You can also use a `for` loop to print or examine each character without range.

```
major = "CSc"
for letter in major:
    print(letter)
```

Output:

```
C
S
c
```

String tests

Method	Description
`startswith(`**str**`)`	whether one contains other's characters at start
`endswith(`**str**`)`	whether one contains other's characters at end

```
name = "Voldermort"
if name.startswith("Vol"):
    print("He who must not be named")
```

- The `in` keyword can be used to test if a string contains another string.

  example: `"er" in name` **# true**

String question

- A *Caesar cipher* is a simple encryption where a message is encoded by shifting each letter by a given amount.
  - e.g. with a shift of 3, A → D, H → K, X → A, and Z → C

- Write a program that reads a message from the user and performs a Caesar cipher on its letters:

```
Your secret message: Brad thinks Angelina is cute
Your secret key: 3
The encoded message: eudg wklqnv dqjholqd lv fxwh
```

`Strings` and `ints`

- All `char` values are assigned numbers internally by the computer, called *ASCII* values.

  - Examples:
    `'A'` is 65, `'B'` is 66, `' '` is 32
    `'a'` is 97, `'b'` is 98, `'*'` is 42

  - One character long `Strings` and `ints` can be converted to each other
    `ord('a')` is 97, `chr(103)` is 'g'

  - This is useful because you can do the following:
    `chr(ord('a' + 2))` is `'c'`

Cumulative Algorithms

Adding many numbers

- How would you find the sum of all integers from 1-1000?

```
# This may require a lot of typing
sum = 1 + 2 + 3 + 4 + ...
print("The sum is", sum)
```

- What if we want the sum from 1 - 1,000,000?
  Or the sum up to any maximum?
  - How can we generalize the above code?

Cumulative sum loop

```
sum = 0
for i in range(1, 1001):
    sum = sum + i

print("The sum is", sum)
```

- **cumulative sum**: A variable that keeps a sum in progress and is updated repeatedly until summing is finished.
  - The `sum` in the above code is an attempt at a cumulative sum.
  - Cumulative sum variables must be declared *outside* the loops that update them, so that they will still exist after the loop.

Cumulative product

- This cumulative idea can be used with other operators:

```
product = 1
for i in range(1, 21):
    product = product * 2

print("2 ^ 20 =", product)
```

- How would we make the base and exponent adjustable?

`input` and cumulative sum

- We can do a cumulative sum of user input:

```
sum = 0
for i in range(1, 101):
    next = int(input("Type a number: "))
    sum = sum + next

print("The sum is", sum)
```

Cumulative sum question

- Modify the `receipt` program from lecture 2
  - Prompt for how many people, and each person's dinner cost.
  - Use functions to structure the solution.

- Example log of execution:

```
How many people ate? 4
Person #1: How much did your dinner cost? 20.00
Person #2: How much did your dinner cost? 15
Person #3: How much did your dinner cost? 30.0
Person #4: How much did your dinner cost? 10.00

Subtotal: $75.0
Tax: $6.0
Tip: $11.25
Total: $92.25
```

Cumulative sum answer

```
# This program enhances our Receipt program using a cumulative sum.
def main():
    subtotal = meals()
    results(subtotal)

# Prompts for number of people and returns total meal subtotal.
def meals():
    people = float(input("How many people ate? "))
    subtotal = 0.0;            # cumulative sum

    for i in range(1, people + 1):
        person_cost = float(input("Person #" + str(i) +
                       ": How much did your dinner cost? "))
        subtotal = subtotal + person_cost  # add to sum
    return subtotal
...
```

Cumulative answer, cont'd.

```
# Calculates total owed, assuming 8% tax and 15% tip
def results(subtotal):
    tax = subtotal * .08
    tip = subtotal * .15
    total = subtotal + tax + tip

    print("Subtotal: $" + str(subtotal))
    print("Tax: $" + str(tax))
    print("Tip: $" + str(tip))
    print("Total: $" + str(total))
```

Opening Business

- Topics Covered
  - Cumulative algorithms

- Test next week!

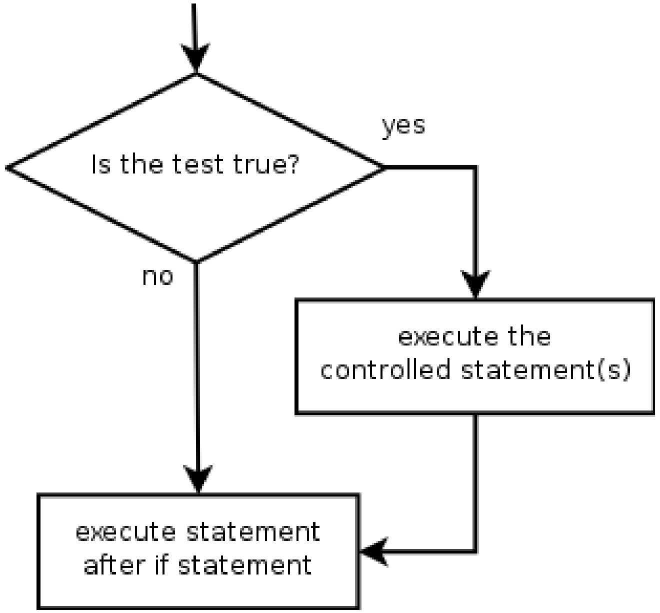
- Office Hours:
  - Monday-Thursday 8:30-10am
  - Lots of TA hours! Check Discord and BBLearn!

The `if/else` statement

The `if` statement

Executes a block of statements only if a test is true

```
if test:
    statement
    ...
    statement
```


Is the test true?
yes
no
execute the controlled statement(s)
execute statement after if statement

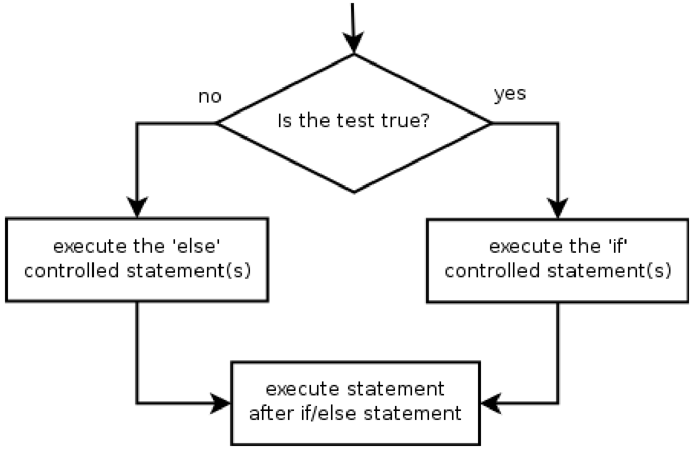
- Example:

```
gpa = float(input("gpa? "))
if gpa >= 2.0:
    print("Application accepted.")
```

The `if/else` statement

Executes one block if a test is true, another if false

```
if test:
    statement(s)
else:
    statement(s)
```


no
Is the test true?
yes
execute the 'else' controlled statement(s)
execute the 'if' controlled statement(s)
execute statement after if/else statement

- Example:

```
gpa = float(input("gpa? "))
if gpa >= 2.0:
    print("Welcome to Mars University!")
else:
    print("Application denied.")
```

Relational expressions

- `if` statements use logical tests.

```
if i <= 10: ...
```

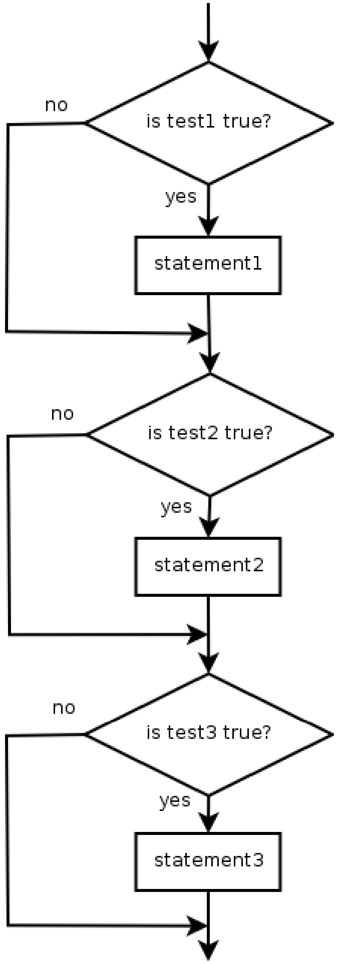
  - These are `boolean` expressions
- Tests use *relational operators*:

Operator	Meaning	Example	Value
`==`	equals	`1 + 1 == 2`	`True`
`!=`	does not equal	`3.2 != 2.5`	`True`
`<`	less than	`10 < 5`	`False`
`>`	greater than	`10 > 5`	`True`
`<=`	less than or equal to	`126 <= 100`	`False`
`>=`	greater than or equal to	`5.0 >= 5.0`	`True`

Misuse of `if`

- What's wrong with the following code?

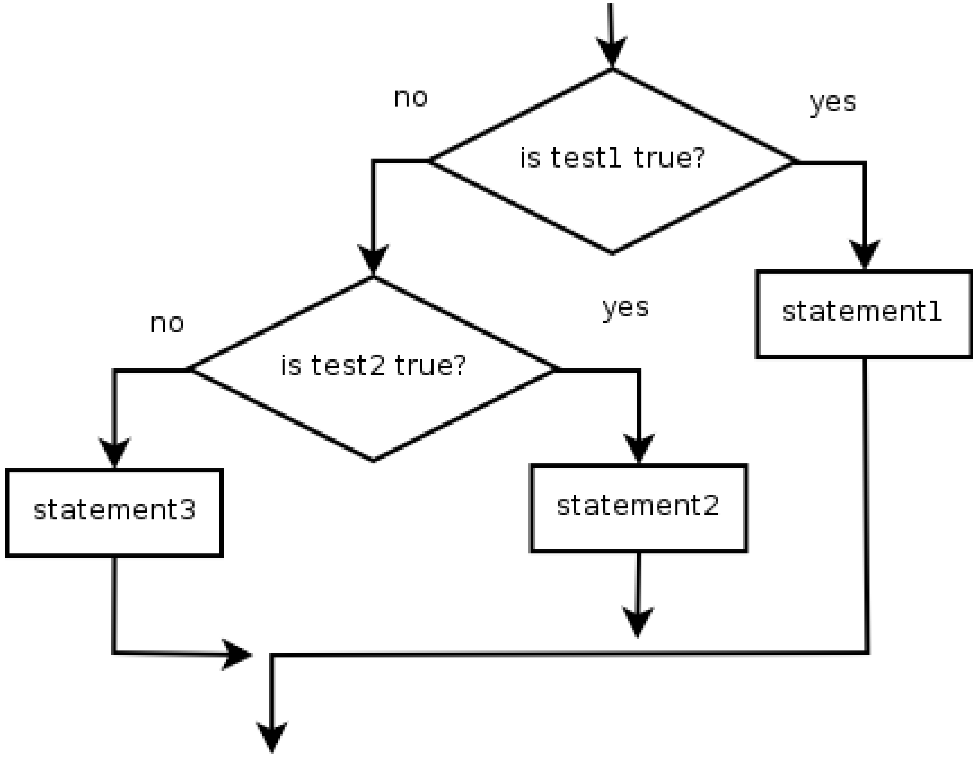
```
percent = float(input("What percentage did you earn? "))
if percent >= 90:
    print("You got an A!")
if percent >= 80:
    print("You got a B!")
if percent >= 70:
    print("You got a C!")
if percent >= 60:
    print("You got a D!")
if percent < 60:
    print("You got an F!")
...
```



Nested `if/else`

Chooses between outcomes using many tests

```
if test:
    statement(s)
elif test:
    statement(s)
else:
    statement(s)
```



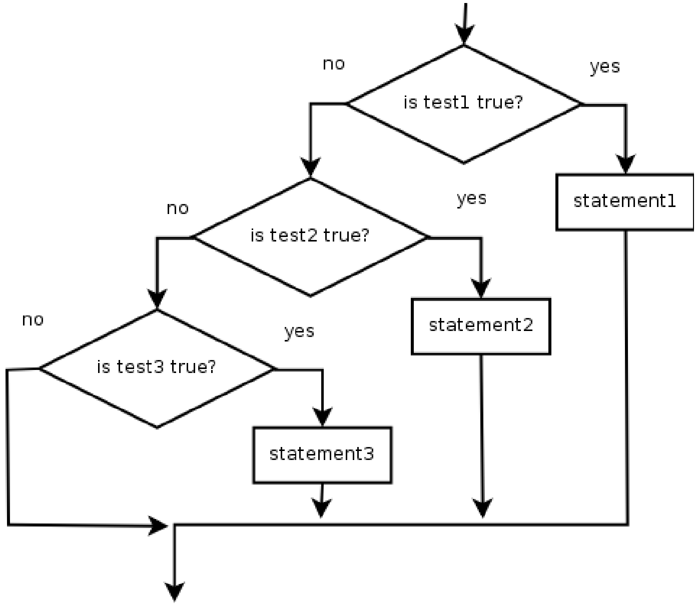
- Example:

```
if x > 0:
    print("Positive")
elif x < 0:
    print("Negative")
else:
    print("Zero")
```

Nested `if/elif/elif`

- If it ends with `else`, exactly one path must be taken.
- If it ends with `if`, the code might not execute any path.

```
if test:
    statement(s)
elif test:
    statement(s)
elif test:
    statement(s)
```



- Example:

```
if place == 1:
    print("Gold medal!")
elif place == 2:
    print("Silver medal!")
elif place == 3:
    print("Bronze medal.")
```

Nested `if` structures

- exactly 1 path *(mutually exclusive)*

```
if test:
    statement(s)
elif test:
    statement(s)
else:
    statement(s)
```

- 0 or 1 path *(mutually exclusive)*

```
if test:
    statement(s)
elif test:
    statement(s)
elif test:
    statement(s)
```

- 0, 1, or many paths *(independent tests; not exclusive)*

```
if test:
    statement(s)

if test:
    statement(s)

if test:
    statement(s)
```

Which nested `if/else`?

- **(1) if/if/if (2) nested if/else (3) nested if/elif/elif**
  - Whether a user is lower, middle, or upper-class based on income.
    - **(2)** nested `if / elif / else`
  - Whether you made the dean's list (GPA ≥ 3.8) or honor roll (3.5-3.8).
    - **(3)** nested `if / elif`
  - Whether a number is divisible by 2, 3, and/or 5.
    - **(1)** sequential `if / if / if`
  - Computing a grade of A, B, C, D, or F based on a percentage.
    - **(2)** nested `if / elif / elif / elif / else`

Nested `if/else` question

Write a program that produces output like the following:

```
This program reads data for two
people and computes their basal
metabolic rate and burn rate.

Enter next person's information:
height (in inches)? 73.5
weight (in pounds)? 230
age (in years)? 35
gender (male or female)? male

Enter next person's information:
height (in inches)? 71
weight (in pounds)? 220.5
age (in years)? 20
gender (male or female)? female

Person #1 basal metabolic rate = 2042.3
high resting burn rate
Person #2 basal metabolic rate = 1868.4
moderate resting burn rate
```

- Basal Metabolic Rate Formula:

male BMR = 4.54545 x (weight in lb) + 15.875 x (height in inches) - 5 x (age in years) + 5

female BMR = 4.54545 x (weight in lb) + 15.875 x (height in inches) - 5 x (age in years) - 161

BMR	Burn Level
below 12000	low
1200 to 2000	moderate
above 2000	high

Nested `if/else` answer

```
# This program finds the basal metabolic rate (BMR) for two
# individuals. This variation includes several functions
# other than main.

# introduces the program to the user
def give_intro():
    print("This program reads data for two")
    print("people and computes their basal")
    print("metabolic rate and burn rate.")
    print()

# prompts for one person's statistics, returning the BMI
def get_bmr(person):
    print("Enter person", person, "information:")
    height = float(input("height (in inches)? "))
    weight = float(input("weight (in pounds)? "))
    age = float(input("age (in years)? "))
    gender = input("gender (male or female)? ")
    bmr = bmr_for(height, weight, age, gender)
    print()
    return bmr
...
```

Nested `if/else`, cont'd.

```
# this function contains the basal metabolic rate formula for
# converting the given height (in inches), weight
# (in pounds), age (in years) and gender (male or female) into a BMR
def bmr_for(height, weight, age, gender):
    bmr = 4.54545 * weight + 15.875 * height - 5 * age
    if gender.lower() == "male":
        bmr += 5
    else:
        bmr -= 161
    return bmr

# reports the overall bmr values and status
def report_results(bmr1, bmr2):
    print("Person #1 basal metabolic rate =", round(bmr1, 1))
    report_status(bmr1)
    print("Person #2 basal metabolic rate =", round(bmr2, 1))
    report_status(bmr2)

# reports the burn rate for the given BMR value
def report_status(bmr):
    if bmr < 1200:
        print("low resting burn rate");
    elif bmr <= 2000:
        print("moderate resting burn rate")
    else: # bmr1 > 2000
        print("high resting burn rate")

def main():
    give_intro()
    bmr1 = get_bmr(1)
    bmr2 = get_bmr(2)
    print(bmr1, bmr2)
    report_results(bmr1, bmr2)

main()
```

Factoring `if/else` code

- **factoring**: Extracting common/redundant code.
  - Can reduce or eliminate redundancy from `if/else` code.
- Example:

```
if a == 1:
    print(a)
    x = 3
    b = b + x
elif a == 2:
    print(a)
    x = 6
    y = y + 10
    b = b + x
else:        # a == 3
    print(a)
    x = 9
    b = b + x
```

```
print(a)
x = 3 * a
if a == 2:
    y = y + 10
b = b + x
```

Exam 1 – February 21 & 23

- Quiz and Homework will be review

- They are due BEFORE the exam: Sunday.

- Material covered will include chapters 1-4 of ‘Building Python Programs’ and/or the material discuss in class through Feb 3rd.

- Some questions will require programming

Wrap Up

- Topics Covered
  - More about strings
  - Cumulative algorithms
  - If/elif/else

- Test next week!

- Office Hours:
  - 8:30-10am Mon - Thurs