Data Driven World
Week 1

Simon Perrault– Singapore University of Technology and Design

Instructor

- Natalie AGUS
- Lecturer
- Information Systems Technology and Design (ISTD) Pillar
- PhD from SUTD
- Email: natalie_agus@sutd.edu.sg
- Office: 1.502.25
- Office Hours: By appointment, contact via Telegram or MS Teams Chat

A quick word about me

- Simon PERRAULT
- Assistant Professor
- Information Systems Technology and Design (ISTD) pillar
- PhD from Telecom ParisTech (Paris, France)
- Email: simon_perrault@sutd.edu.sg
- Office: 1.502.20
- Office Hours: Tuesdays, 1:30pm - 3pm (via MS Teams)

Important Points

- Attendance is mandatory
  - We will take attendance each class
- COVID Guidelines:
  - Pictures of everyone in class
  - No food/drink in class

Objectives of this course

- Analyse different algorithms' complexity in terms of computation time using Python computational model
- Identify recursive structure in a problem and implement its solution in Python
- Explain UML diagrams and design software using basic UML diagrams
- Apply appropriate data structure and implement them using object oriented design
- Implement algorithm to find coefficients for linear regression by minimizing its error
- Implement algorithm to classify categorical data using logistic regression for binary category and above
- Analyse and evaluate linear regression using mean square error and correlation coefficient
- Analyse and evaluate logistic regression using confusion matrix, its accuracy and recall
- Design state machine and implement it using object oriented paradigm
- Fix syntax errors and debug semantic errors using print statements

Submission Policy and Plagiarism

- **You will do the assignment on your own and will not copy paste solutions from someone else.**
- **You will not post any solutions related to this course to a private/public repository that is accessible by the public/others.**
- Students are allowed to have a **private repository** for their cohort and homework problem sets which no one can access.
- For mini projects, students can only invite their partners as **collaborators** to a private repository.
- Failing to follow the Code of Honour will result in failing the course and/or being submitted to the University Disciplinary Committee. The consequences apply to both the person who shares their work and the person who copies the work.

Topics

- Day 1S1: Getting started, key concepts, configuring and installing Python
- Day 1S1b: Variables, math operators, comments, printing and getting
- Day 1S2: None type, Boolean types and functions
- Day 1S3: If, elif, else, while, break statements
- Day 2S1: For loops, generators
- Day 2S2: The list type
- Day 2S3: Imports and quick intro to some important libraries
- Day 3S1: Everything about strings
- Day 3S2: Tuples, sets, dictionaries and object-oriented thinking

MS Teams + Course Handout

- Are you on MS Teams? If not, join now.
- Course Handout
  - Everything you need to know about the class
  - https://docs.google.com/document/d/1Gxt9sqX4mpC6pkRyIsG6PloBFYQujkAQS9obGJee2BM/edit?usp=sharing

Week 1 - Python

Quick Python Recap

Variables

- Used to store values

```
# Declaring a variable x with a value of 9
x = 9
# Updating x to 11
x = 11
# Declaring a variable name with my name in it
name = “Simon”
```

Python basic operators

By default, Python comes with a few **basic math operators, for number (int/float) variables**.

Python basic operators

By default, Python comes with a few **basic math operators, for number (int/float) variables**.

- **+:** addition
- **-:** subtraction
- ***:** multiplication

Basic math operators

```
1 # Two numbers
2 a, b = 5, 2
3 print(a)
4 print(b)
```

```
5
2
```

```
1 # Addition
2 print(a+b)
```

```
7
```

```
1 # Subtraction
2 print(a-b)
```

```
3
```

```
1 # Multiplication
2 print(a*b)
```

```
10
```

Python basic operators

By default, Python comes with a few **basic math operators, for number (int/float) variables**.

- **+:** addition
- **-:** subtraction
- ***:** multiplication
- **/:** division
- **//:** integer division
- **%:** remaining of the integer division (a.k.a. modulus)
- ****:** exponentiation

Basic math operators

```
# Two numbers
a, b = 5, 2
print(a)
print(b)
```

```
5
2
```

```
# Addition
print(a+b)
```

```
7
```

```
# Subtraction
print(a-b)
```

```
3
```

```
# Multiplication
print(a*b)
```

```
10
```

```
# Division
print(a/b)
```

```
2.5
```

```
# Floor division
print(a//b)
```

```
2
```

```
# Modulus
print(a%b)
```

```
1
```

```
# Exponentiation
print(a**b)
```

```
25
```

Python basic operators

By default, Python comes with a few **basic math operators, for number (int/float) variables**.

- **+:** addition
- **-:** subtraction
- ***:** multiplication
- **/:** division
- **//:** integer division
- **%:** remaining of the integer division (a.k.a. modulus)
- ****:** exponentiation

Basic math operators

```
# Two numbers
a, b = 5, 2
print(a)
print(b)
```

```
5
2
```

```
# Addition
print(a+b)
```

```
7
```

```
# Subtraction
print(a-b)
```

```
3
```

```
# Multiplication
print(a*b)
```

```
10
```

```
# Division
print(a/b)
```

```
2.5
```

```
# Floor division
print(a//b)
```

```
2
```

```
# Modulus
print(a%b)
```

```
1
```

```
# Exponentiation
print(a**b)
```

```
25
```

Python basic operators

By default, Python comes with a few **basic math operators, for number (int/float) variables**.

- **+:** addition
- **-:** subtraction
- ***:** multiplication
- **/:** division
- **//:** integer division
- **%:** remaining of the integer division (a.k.a. modulus)
- ****:** exponentiation

Watch out for the types of variables!

```
1 a = 2
2 b = "Hello World!"
3 print(a+b)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-4eaf3ef22f17> in <module>
      1 a = 2
      2 b = "Hello World!"
----> 3 print(a+b)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Combining operators and assignments

You can mix both operations and assignments.

Combining math operations and assignments

```
1  # Addition
2  a, b = 5, 2
3  print(a)
4  a += b
5  print(a)
```

```
5
7
```

```
1  # Multiplication
2  a, b = 5, 2
3  print(a)
4  a *= b
5  print(a)
```

```
5
10
```

Combining operators and assignments

You can mix both operations and assignments.

- For instance, the **operation +=** sums values and the right- and left-hand sides, and assigns the result to the variable on the left-hand side.

Combining math operations and assignments

```
1  # Addition
2  a, b = 5, 2
3  print(a)
4  a += b
5  print(a)
```

```
5
7
```

```
1  # Multiplication
2  a, b = 5, 2
3  print(a)
4  a *= b
5  print(a)
```

```
5
10
```

Combining operators and assignments

You can mix both operations and assignments.

- For instance, the **operation +=** sums values and the right- and left-hand sides, and assigns the result to the variable on the left-hand side.

- Similarly, we have: **-=, *=, /=, //=, %=, **=.**

Combining math operations and assignments

```
1  # Addition
2  a, b = 5, 2
3  print(a)
4  a += b
5  print(a)
```

```
5
7
```

```
1  # Multiplication
2  a, b = 5, 2
3  print(a)
4  a *= b
5  print(a)
```

```
5
10
```

Boolean comparisons

- **Definition (the == operator):**
  The == operator is used to check if two variables have identical values.
  The result of this operation is a Boolean, with value
  - **True**, if both variables have **identical values**;
  - and **False**, **otherwise**.
- **Note:** Also, works if variables have mixed int/float types.

```
a = 1.0
b = 1
c = 2
bool1 = (a == b)
bool2 = (a == c)
print(type(a))
print(type(b))
print(type(c))
print(bool1)
print(bool2)
```

```
<class 'float'>
<class 'int'>
<class 'int'>
True
False
```

Boolean comparisons

- **Definition (the == operator):**
  The == operator is used to check if two variables have identical values.
  The result of this operation is a Boolean, with value
  - **True**, if both variables have **identical values**;
  - and **False**, **otherwise**.
- **Note:** Also, works if variables have mixed int/float types.
  But **not with str and int/float!**

```
a = 1.0
b = 1
c = 2
bool1 = (a == b)
bool2 = (a == c)
print(type(a))
print(type(b))
print(type(c))
print(bool1)
print(bool2)
```

```
<class 'float'>
<class 'int'>
<class 'int'>
True
False
```

```
a = "1"
b = 1.0
c = 1
bool1 = (a == b)
bool2 = (a == c)
print(type(a))
print(type(b))
print(type(c))
print(bool1)
print(bool2)
```

```
<class 'str'>
<class 'float'>
<class 'int'>
False
False
```

Boolean comparisons

- Following the same logic as the == operator, we can define, the following operators, **for numerical types (int/float).**

```
a = 1
b = 1
c = 2
bool1 = (a != b)
bool2 = (a != c)
print(bool1)
print(bool2)
```

```
False
True
```

```
a = "1"
b = 1
c = 1.0
bool1 = (a != b)
bool2 = (a != c)
bool3 = (b != c)
print(bool1)
print(bool2)
print(bool3)
```

```
True
True
False
```

Boolean comparisons

- Following the same logic as the == operator, we can define some additional operators.

- **!=:** if variables have **different values**.

```
a = 1
b = 1
c = 2
bool1 = (a != b)
bool2 = (a != c)
print(bool1)
print(bool2)
```

```
False
True
```

```
a = "1"
b = 1
c = 1.0
bool1 = (a != b)
bool2 = (a != c)
bool3 = (b != c)
print(bool1)
print(bool2)
print(bool3)
```

```
True
True
False
```

Boolean comparisons

- Following the same logic as the == operator, we can define some additional operators.

- **!=:** if variables have **different values**.

- **Note:** careful with the types on both sides! Let us only use these operators with similar types on both sides!

```
a = 1
b = 1
c = 2
bool1 = (a != b)
bool2 = (a != c)
print(bool1)
print(bool2)
```

```
False
True
```

```
a = "1"
b = 1
c = 1.0
bool1 = (a != b)
bool2 = (a != c)
bool3 = (b != c)
print(bool1)
print(bool2)
print(bool3)
```

```
True
True
False
```

Boolean comparisons

Similarly,

- **>:** if variable on the left-hand side has a **higher numerical value**, than the one on the right-hand side.
- **<:** same as **>**, but checking for **lower numerical value**.

```
a = 1
b = 1
c = 2
bool1 = (a > b)
bool2 = (a > c)
print(bool1)
print(bool2)
```

```
False
False
```

```
a = 1
b = 1
c = 2
bool1 = (a < b)
bool2 = (a < c)
print(bool1)
print(bool2)
```

```
False
True
```

Boolean comparisons

Similarly,

- **>:** if variable on the left-hand side has a **higher numerical value**, than the one on the right-hand side.
- **<:** same as **>**, but checking for **lower numerical value**.

- **Note:** careful with the types on both sides! Let us only use these operators with similar types!

```
a = "1"
b = 1
print(type(a))
print(type(b))
bool1 = (a > b)
print(bool1)
```

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

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-22-dedd4801f811> in <module>
      3 print(type(a))
      4 print(type(b))
----> 5 bool1 = (a > b)
      6 print(bool1)

TypeError: '>' not supported between instances of 'str' and 'int'
```

Boolean comparisons

Similarly,

- **>=:** if variable on the left-hand side has a **higher or equal numerical value**, than the one on the right-hand side.
- **<=:** same as **>=**, but checking for **lower or equal numerical value**.

- **Note:** careful with the types on both sides! Let us only use these operators with similar types!

```
a = 1
b = 1
c = 2
bool1 = (a >= b)
bool2 = (a >= c)
print(bool1)
print(bool2)
```

```
True
False
```

```
a = 1
b = 1
c = 2
bool1 = (a <= b)
bool2 = (a <= c)
print(bool1)
print(bool2)
```

```
True
True
```

The if statement

The **if** statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.
  - If the Boolean condition is **False**, ignore the block of code in the **if** statement.
  - Once we are done executing the code in **if** (or ignoring it), move on to the next (non-indented) line.

```
condition = True # or False value
if(condition):
    print("This will be printed if condition is set to True.")
    print("It will not print if condition is set to False.")
print("This will be printed: not indented, outside of the if statement.")
```

```
This will be printed if condition is set to True.
It will not print if condition is set to False.
This will be printed: not indented, outside of the if statement.
```

```
condition = False # or True value
if(condition):
    print("This will be printed if condition is set to True.")
    print("It will not print if condition is set to False.")
print("This will be printed: not indented, outside of the if statement.")
```

```
This will be printed: not indented, outside of the if statement.
```

The elif statement

How it works:

- If the Boolean in the **if** statement is **True**, execute the code inside the **if**, ignore the **elif**.
- **Otherwise,** check for the Boolean in **elif**, and execute the code indented inside the **elif**, if this second Boolean condition is **True**. Otherwise, ignore it.

```
# Some booleans
bool1 = False
bool2 = True
# If statement, with False boolean condition
if(bool1):
    print("1. This will NOT be printed, because bool1 is False.")
# Elif statement, with True boolean condition
elif(bool2):
    print("2. This will be printed, because the first if block was not executed and bool2 is True.")
```

```
2. This will be printed, because the first if block was not executed and bool2 is True.
```

An example of if/elif code

Example: write a code that receives a number x, and prints one of the following prompts, accordingly:

- "x is strictly positive."
- "x is strictly negative."
- "x is zero."

```
# A number x
x = 10
# An if/elif/else statement
if(x>0):
    print("The number x is strictly positive.")
elif(x<0):
    print("The number x is strictly negative.")
elif(x==0):
    print("The number x is zero.")
```

```
The number x is strictly positive.
```

Let us use the **if/elif** structure to program that!

The else statement (no elif example)

The **else** statement is used to define a block of code to execute, if and only if **ALL** the previous **if/elif** statement have failed.

Same structure as an **elif**, but…

- **Comes last**, after all the **if/elif** statements.
- **No Boolean condition** to be checked.

```
1 bool1 = True
2 if(bool1):
3     print("1. This will be printed, because bool1 is True.")
4 else:
5     print("2. This will NOT be printed, because the previous block was executed.")
```

```
1. This will be printed, because bool1 is True.
```

```
1 bool1 = False
2 if(bool1):
3     print("1. This will NOT be printed, because bool1 is False.")
4 else:
5     print("2. This will be printed, because none of the previous blocks were executed.")
```

```
2. This will be printed, because none of the previous blocks were executed.
```

If-Else Question

- Multiple Choice: What value of x will have Banana displayed?

A. 6

B. 8

C. 10

D. 12

E. It will not be possible

```
if(x > 7):
    print("Apple")
elif(x > 10):
    print("Banana")
else:
    print("Chiku")
```

To answer go to www.socrative.com
Click “Login” > “Student Login”
Room name is PERRAULT6086

The **while** statement

The **while** statement is another type of **conditional structure.**

The while statement

The **while** statement is another type of **conditional structure.**

The **if** statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.
  - If the Boolean condition is not **True**, ignore the block of code in the **if** statement.
  - Once we are done executing the code in **if** (or ignoring it), move on to the next (non-indented) line.

The while statement

The **while** statement is another type of **conditional structure.**

- **How it works:**
  - If the Boolean condition specified for the **while** statement is **True**, then execute the block of code inside the **while** statement.
  - If the Boolean condition **False**, ignore the block of code in the **while** statement.

The **if** statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.
  - If the Boolean condition is **False**, ignore the block of code in the **if** statement.
  - Once we are done executing the code in **if** (or ignoring it), move on to the next (non-indented) line.

The while statement

The **while** statement is another type of **conditional structure.**

- **How it works:**
  - If the Boolean condition specified for the **while** statement is **True**, then execute the block of code inside the **while** statement.
  - If the Boolean condition is **False**, ignore the block of code in the **while** statement.
  - Once we are done executing the code in **while**, **move back to the while statement, and repeat until the condition is no longer True.**

The **if** statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.
  - If the Boolean condition is **False**, ignore the block of code in the **if** statement.
  - Once we are done executing the code in **if** (or ignoring it), **move on to the next (non-indented) line.**

The while statement

The **while** statement is another type of **conditional structure.**

- **How it works:**
  - If the Boolean condition specified for the **while** statement is **True**, then execute the block of code inside the **while** statement.
  - If the Boolean condition is **False**, ignore the block of code in the **while** statement.
  - Once we are done executing the code in **while**, **move back to the while statement, and repeat until the condition is no longer True.**

```
1  # Counting from 1 to 10
2  x = 0
3  print("Counting from 1 to 10...")
4  while(x<10):
5      x = x + 1
6      print(x)
7  print("Done!")
```

```
Counting from 1 to 10...
1
2
3
4
5
6
7
8
9
10
Done!
```

Loops Question

Multiple Choice: What is printed by the following code? Output is on one line to save space.

A. 6 5
B. 6 5 4
C. 5 4
D. 5 4 3
E. 6 5 4 3

```
x = 6
while x > 4:
    x = x - 1
    print(x, end=" ")
```

To answer go to www.socrative.com
Click "Login" > "Student Login"
Room name is PERRAULT6086

The for statement

- Sometimes in programming, there is a block of code that you want to repeat for a fixed number of times.
- It could be done with a **while** statement, but there is a more convenient way.

```
1  # Counting from 1 to 5
2  # (While loop edition)
3  i = 0
4  while(i<5):
5      i += 1
6      print(i)
```

```
1
2
3
4
5
```

The for statement

- Sometimes in programming, there is a block of code that you want to repeat for a fixed number of times.
- It could be done with a **while** statement, but there is a more convenient way.
- **More convenient way:** the **for** statement is used to repeat a given block of code **for** a given number of times.

```
1  # Counting from 1 to 5
2  # (While loop edition)
3  i = 0
4  while(i<5):
5      i += 1
6      print(i)
```

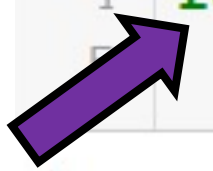
```
1
2
3
4
5
```

The for statement

The for loop - How it works:

- Use the **for** keyword,

```
1  # Counting from 1 to 5
2  # (For loop edition)
3  my_list = [1, 2, 3, 4, 5]
4  for i in my_list:
       print(i)
```



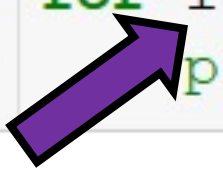
```
1
2
3
4
5
```

The for statement

The for loop - How it works:

- Use the **for** keyword,
- It is immediately followed by a **variable name**, called an **iteration variable**,

```
1  # Counting from 1 to 5
2  # (For loop edition)
3  my_list = [1, 2, 3, 4, 5]
4  for i in my_list:
5      print(i)
```



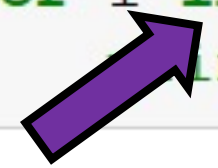
```
1
2
3
4
5
```

The for statement

The for loop - How it works:

- Use the **for** keyword,
- It is immediately followed by a **variable name**, called an **iteration variable**,
- Use the **in** keyword to indicate that the **iteration variable** will take values in a given **list**,

```
1  # Counting from 1 to 5
2  # (For loop edition)
3  my_list = [1, 2, 3, 4, 5]
4  for i in my_list:
5      print(i)
```

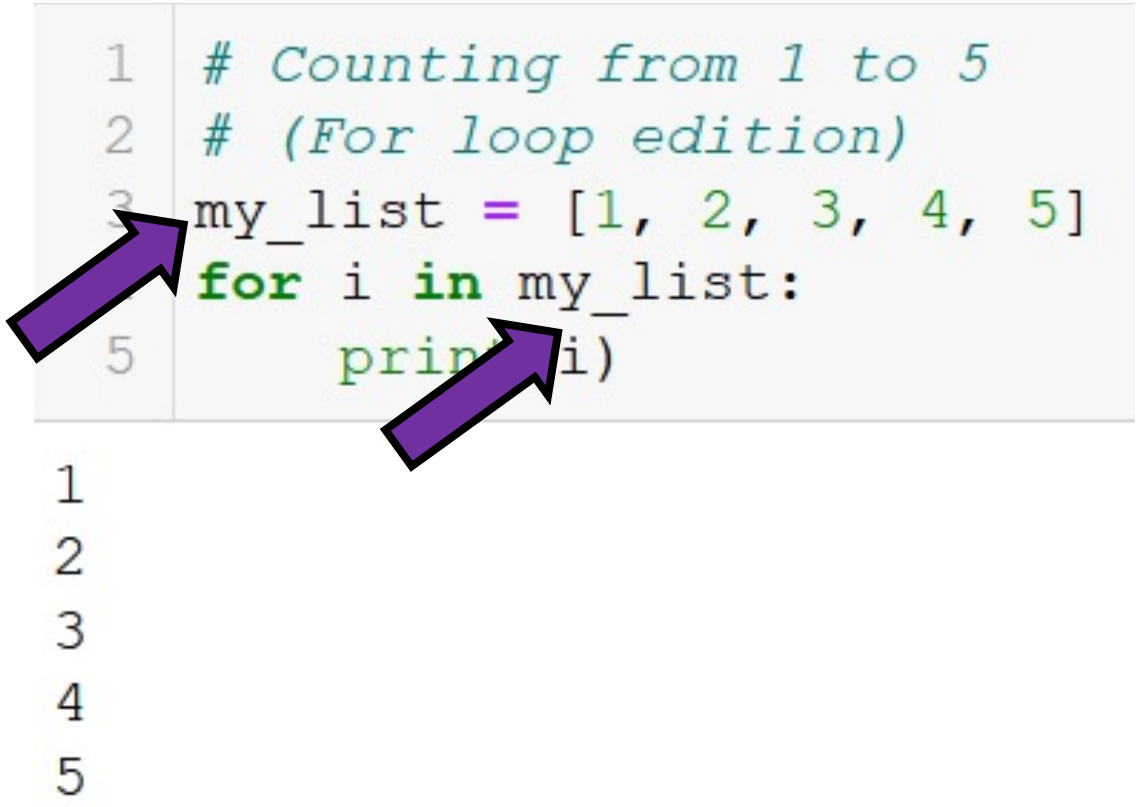


```
1
2
3
4
5
```

The for statement

The for loop - How it works:

- Use the **for** keyword,
- It is immediately followed by a **variable name**, called an **iteration variable**,
- Use the **in** keyword to indicate that the **iteration variable** will take values in a given **list**,
- Provide a **list** object, finish with a **:** **symbol.**

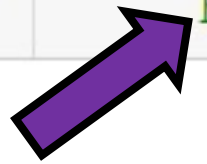


The for statement

The for loop - How it works:

- Use the **for** keyword,
- It is immediately followed by a **variable name**, called an **iteration variable**,
- Use the **in** keyword to indicate that the **iteration variable** will take values in a given **list**,
- Provide a **list** object, finish with a **: symbol.**
- **Indent some** code to be repeated inside the **for**, as in **if**/**while**.

```
1  # Counting from 1 to 5
2  # (For loop edition)
3  my_list = [1, 2, 3, 4, 5]
4  for i in my_list:
5      print(i)
```



```
1
2
3
4
5
```

The for statement

The for loop - How it works:

- Use the **for** keyword,
- It is immediately followed by a **variable name**, called an **iteration variable**,
- Use the **in** keyword to indicate that the **iteration variable** will take values in a given **list**,
- Provide a **list** object, finish with a **: symbol.**
- **Indent some** code to be repeated inside the **for**, as in **if**/**while**.

```
1  # Counting from 1 to 5
2  # (For loop edition)
3  my_list = [1, 2, 3, 4, 5]
4  for i in my_list:
5      print(i)
```

```
1
2
3
4
5
```

Notice how **the iteration variable value changes after each repetition of the code** inside the for loop.

For Loops - Question

True/False

This code will work?

```
f_series = [1,1,2,3,5,8,11]
for i in f_series:
    print(f_series[i])
```

For Loops - Question

Multiple Choice: For code snippet B to do the same thing as code snippet A, the underlined portion could be

A. range(series)
B. range(len(series))
C. range()
D. len(series)

series = [1,3,6,10,15,21,28]
for i in series:
    print(i)
	series = [1,3,6,10,15,21,28]
	for i in _____ :
	    print(series[i])

To answer go to www.socrative.com
Click “Login” > “Student Login”
Room name is PERRAULT6086

The range() generator

- **Solution:** The **range()** generator can be used to replace the list object in the **for** loop definition.
- It receives an integer **n.**
- Here, **range(n)** means: the **iteration variable** will take **n** successive values, starting from 0 and incrementing by 1 each time.

```
1 my_list = [0,1,2,3,4]
2 for i in my_list:
3     print(i)
```

```
0
1
2
3
4
```

```
1 for j in range(5):
2     print(j)
```

```
0
1
2
3
4
```

The range() iterator

Up to three parameters can be given to the **range()** generator.

- **2 parameters: range(m, n)** makes the **iteration variable** take successive values, starting from **m (instead of 0)** and incrementing by 1 each time, until we reach **n (n not included)**.
- **3 parameters: range(m, n, p)** makes the **iteration variable** take successive values, starting from **m (instead of 0)** and incrementing by **p (instead of 1)** each time, until we reach **n (n not included)**.

```
1 for j in range(-1, 6):
2     print(j)
```

```
-1
0
1
2
3
4
5
```

```
1 for j in range(1, 9, 2):
2     print(j)
```

```
1
3
5
7
```

The range() iterator

Up to three parameters can be given to the **range()** iterator.

- **2 parameters: range(m, n)** makes the **iteration variable** take successive values, starting from **m (instead of 0)** and incrementing by 1 each time, until we reach **n (n not included)**.
- **3 parameters: range(m, n, p)** makes the **iteration variable** take successive values, starting from **m (instead of 0)** and incrementing by **p (instead of 1)** each time, until we reach **n (n not included)**.

```
1 for j in range(-1, 6):
2     print(j)
```

```
-1
0
1
2
3
4
5
```

```
1 for j in range(1, 9, 2):
2     print(j)
```

```
1
3
5
7
```

```
1 for j in range(10, -4, -2):
2     print(j)
```

```
10
8
6
4
2
0
-2
```

Note: if two or more parameters are used, we can play with negative values.

Loops – Final Question

- Multiple Choice: If we execute calculate_sum_of_squares (10) how many lines will be printed out?

A. 0

B. 1

C. 9

D. 10

```
def calculate_sum_of_squares(iterations):
    total = 0
for i in range(1,iterations):
        total += i*i
        print(total)
        return total
```

To answer go to www.socrative.com
Click “Login” > “Student Login”
Room name is PERRAULT6086