CSCI3100 Project Tutorial 2
Game Design Technology

csci3100@cse.cuhk.edu.hk
Department of Computer Science and Engineering
The Chinese University of Hong Kong

February 06, 2023

Outline

- Introduction to Game Development
- The Pygame Library
- Getting Started with Pygame
- Basic Programming Structure of Pygame
- Demo: Punching the Chimpanzee
- Useful links

- This tutorial only gives an example of game design using Pygame
  - In your project, you are free to use other libraries depending on your choices
- Tutorial code (“hello world” program and the "Punching the Chimpanzee"demo code) is available at:
  - https://github.com/CUHK-ARISE/3100-PJ-TUT-2

How to develop games?

Two approaches for game design:

- Low-level libraries and frameworks:
  - libGDX, OpenGL, SDL, …
- High-level game engines:
  - Unity, Unreal, CryEngine
- Re-develop existing games:
  - RPG Maker, Steam workshop, …

Library Comparisons

	Advantages	Disadvantages
Low-level Libraries	• Fast prototyping • Flexible functionalities • High-performance game	• Inconvenient interfaces • More coding effort • Platform dependent
Game Engines	• Many provided functions (e.g., memory management, lighting, asset loading) • Cross platform	• Steel learning curve • Inconvenient bug fix • Heavy development environment
Re-develop Games	• Simple APIs • Fast development	• Limited flexibility

The Pygame Library

- Designed for python programmers
  - One of the most flexible programming language
  - Started year 2000
- Based on **SDL** (Simple DirectMedia Layer)
  - Cross-platform C library for controlling multimedia
- Why Pygame?
  - The easy-to-use python programming language.
  - Suitable for small games in the course
- We recommend Pygame in the project if you have no experience of game development before

Getting Started with Pygame

If you haven't installed python3 yet, please refer to appendix to install python3

- Install Pygame:

```
> python3 –m pip install –U pygame
```

- Run the demo game:

```
> python3 -m pygame.examples.chimp
```

Getting Started with Pygame

If you haven't installed python3 yet, please refer to appendix to install python3

- In

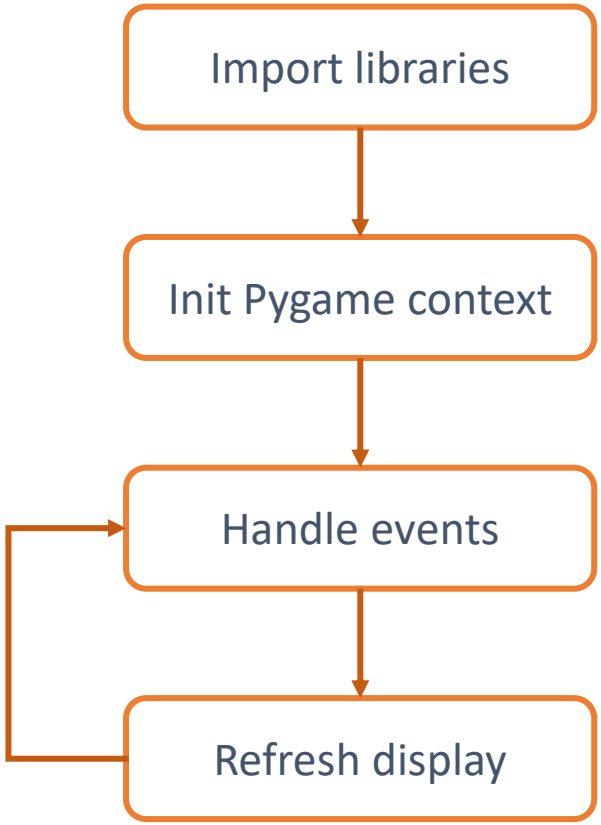
- Ru

Basic Programming Structure

The “Hello world” in Pygame

```python
import pygame as pg
import sys

pg.init()
screen = pg.display.set_mode((1280, 480))
pg.display.set_caption("Punching the Chimpanzee")

while True:
    for event in pg.event.get():
        if event.type == pg.QUIT:
            sys.exit()
    pg.display.update()
```

Typical pygame workflow



Basic Programming Structure

The “Hello world” in Pygame

```python
import pygame as pg
import sys

pg.init()
screen = pg.display.set_mode((1280, 480))
pg.display.set_caption("Punching the Chimpanzee")

while True:
    for event in pg.event.get():
        if event.type == pg.QUIT:
            sys.exit()
    pg.display.update()
```

- Import the pygame libaray and some other system libaraies.
- We use ‘pg’ as an alias of pygame.

Basic Programming Structure

The “Hello world” in Pygame

```python
import pygame as pg
import sys

pg.init()
screen = pg.display.set_mode((1280, 480))
pg.display.set_caption("Punching the Chimpanzee")

while True:
    for event in pg.event.get():
        if event.type == pg.QUIT:
            sys.exit()
    pg.display.update()
```

- `pg.init()`: Initialize the internal modules of pygame.
- Then a screen is setup with 1280px width and 480px height.
- The caption is set to “Punching the Chimpanzee”

Basic Programming Structure

The “Hello world” in Pygame

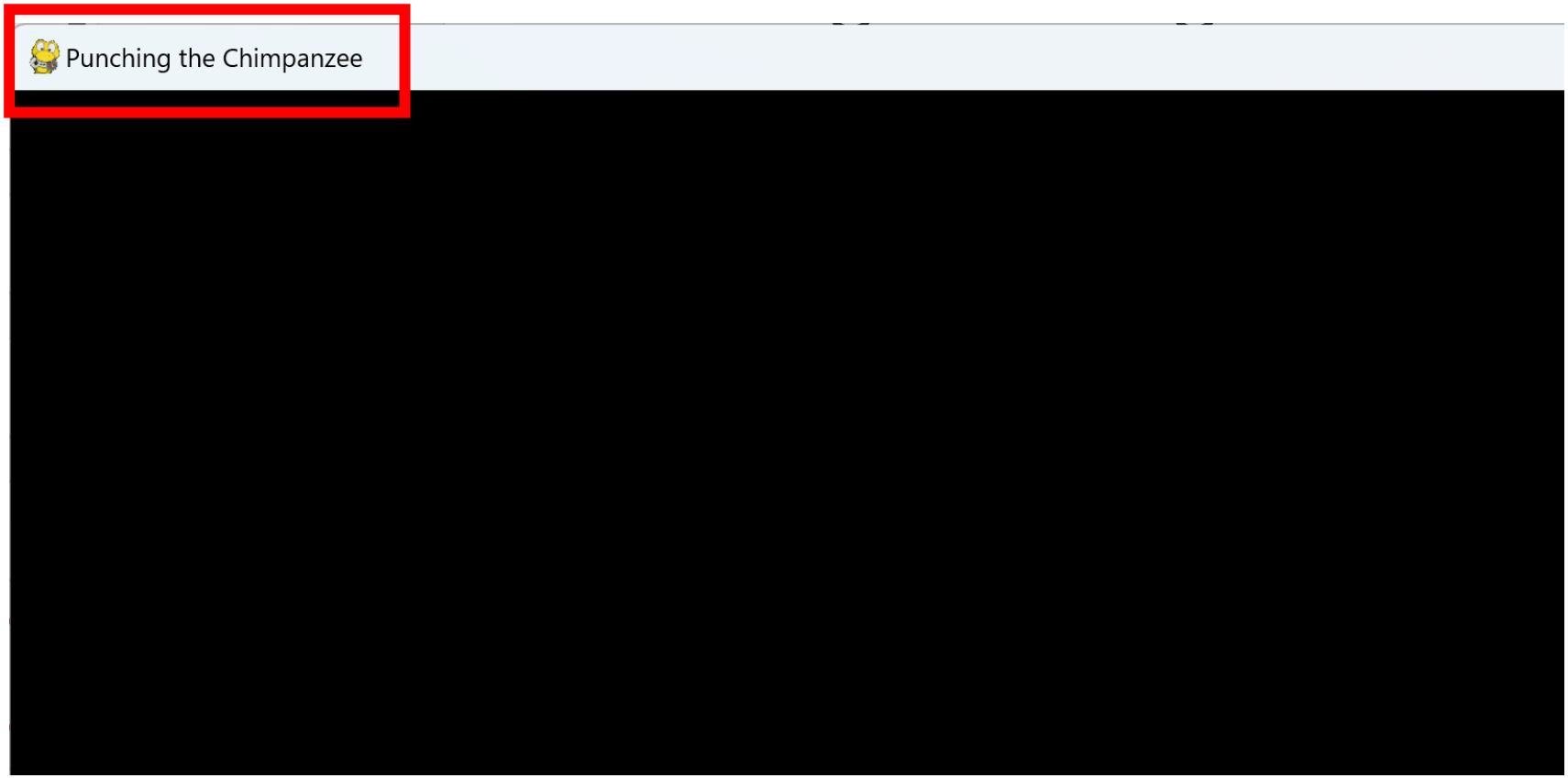
```python
import pygame as pg
import sys

pg.init()
screen = pg.display.set_mode((1280, 480))
pg.display.set_caption("Punching the Chimpanzee")

while True:
    for event in pg.event.get():
        if event.type == pg.QUIT:
            sys.exit()
    pg.display.update()
```

- Event loop of the game.
  - Handle events and update the content in the screen.
- Pygame maintains an event queue and we can get individual events with `event.get()`.
  - E.g., `pg.QUIT` is the event constant representing the quit of the game.

Basic Programming Structure

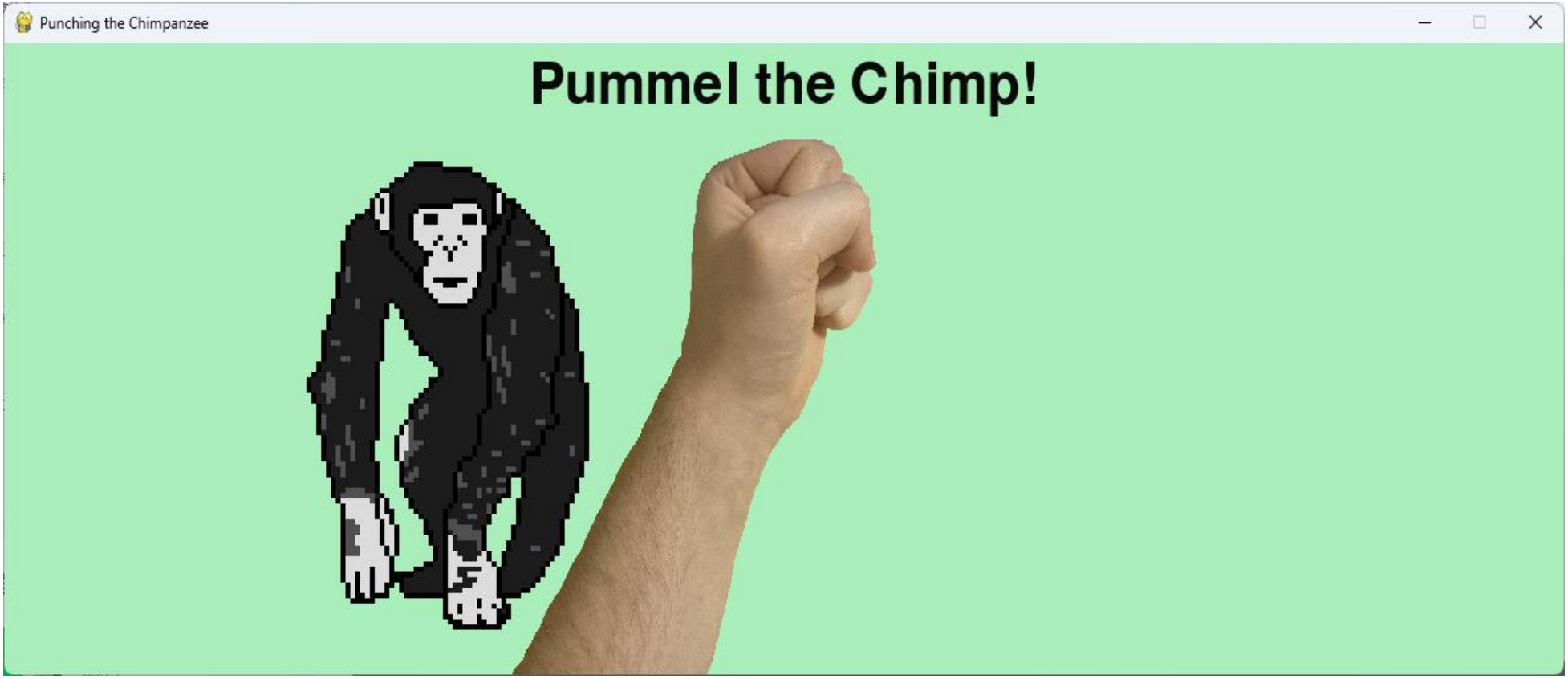
- Run the Pygame “Hello world” program:

```
> python3 basic.py
```



Demo: Punching the Chimpanzee

What we will build…



Demo: Punching the Chimpanzee

```python
import pygame as pg
import os

main_dir = os.path.split(os.path.abspath(__file__))[0]
data_dir = os.path.join(main_dir, 'data')

# functions to load resources
def load_image(name, colorkey=None, scale=1):
    fullname = os.path.join(data_dir, name)
    image = pg.image.load(fullname)

    size = image.get_size()
    size = (size[0] * scale, size[1] * scale)
    image = pg.transform.scale(image, size)

    image = image.convert()
    if colorkey is not None:
        if colorkey == -1:
            colorkey = image.get_at((0, 0))
        image.set_colorkey(colorkey, pg.RLEACCEL)
    return image, image.get_rect()
```

- Import the libraries.
- Prepare the directories for data, i.e., images and sounds effects.

- The load_image function:
  - Get the image with `pg.image.load`.
  - Scale the image with `pg.transform.scale`.
  - Change image color with `image.set_colorkey`.
  - Return the image and its bounding box.

Demo: Punching the Chimpanzee

```python
def load_sound(name):
    class NoneSound:
        def play(self):
            pass

    if not pg.mixer or not pg.mixer.get_init():
        return NoneSound()

    fullname = os.path.join(data_dir, name)
    sound = pg.mixer.Sound(fullname)
    return sound
```

- The `load_sound()` function loads the sound effect of the game.
- Return a `NonSound` class if sound is disabled by the environment.

Demo: Punching the Chimpanzee

```python
class Fist(pg.sprite.Sprite):
    """moves a clenched fist on the screen, following the mouse"""
    def __init__(self) -> None:
        pg.sprite.Sprite.__init__(self)
        self.image, self.rect = load_image("fist.png", -1)
        self.fist_offset = (-235, -80)
        self.punching = False

    def update(self):
        """move the fist based on the mouse position"""
        pos = pg.mouse.get_pos()
        self.rect.topleft = pos
        self.rect.move_ip(self.fist_offset)
        if self.punching:
            self.rect.move_ip(15, 25)

    def punch(self, target):
        """returns true if the fist collides with the target"""
        if not self.punching:
            self.punching = True
            hitbox = self.rect.inflate(-5, -5)
            return hitbox.colliderect(target.rect)

    def unpunch(self):
        """called to pull the fist back"""
        self.punching = False
```

- Derive a class from `pg.sprite.Sprite`, the base class in pygame for visible game objects.
- Overwrite the `update()` method and the `image` and `rect` attribute of Sprite.

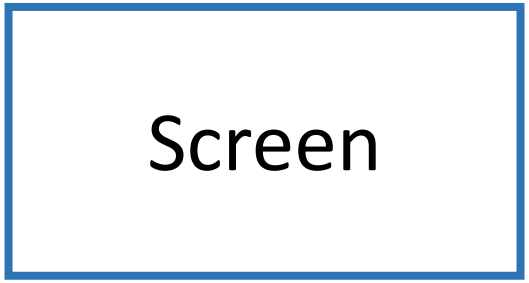
- The Fist class:
  - Move with the mouse.
  - Move a little to the lower right corner when we are punching.

Demo: Punching the Chimpanzee

```python
class Fist(pg.sprite.Sprite):
    """moves a clenched fist on the screen, following the mouse"""
    def __init__(self) -> None:
        pg.sprite.Sprite.__init__(self)
        self.image, self.rect = load_image("fist.png", -1)
        self.fist_offset = (-235, -80)
        self.punching = False

    def update(self):
        """move the fist based on the mouse position"""
        pos = pg.mouse.get_pos()
        self.rect.topleft = pos
        self.rect.move_ip(self.fist_offset)
        if self.punching:
            self.rect.move_ip(15, 25)

    def punch(self, target):
        """returns true if the fist collides with the target"""
        if not self.punching:
            self.punching = True
            hitbox = self.rect.inflate(-5, -5)
            return hitbox.colliderect(target.rect)

    def unpunch(self):
        """called to pull the fist back"""
        self.punching = False
```

Why lower right corner?

(0, 0)



- The Fist class:
  - `punch(self, target):` checks if the fist collides with the target.
  - The hitbox of the fist is smaller than the bounding box of the image.

Demo: Punching the Chimpanzee

```python
class Chimp(pg.sprite.Sprite):
    """moves a monkey critter across the screen. it can spin the
    monkey when it is punched."""
    def __init__(self) -> None:
        pg.sprite.Sprite.__init__(self) # call Sprite initializer
        self.image, self.rect = load_image("chimp.png", -1, 4)
        screen = pg.display.get_surface()
        self.area = screen.get_rect()
        self.rect.topleft = 10, 90
        self.move = 18
        self.dizzy = False

    def update(self):
        if self.dizzy:
            self._spin()
        else:
            self._walk()
```

The Chimp class:

- Derived from `Sprite`.
- Load the image of chimp.
- Define a finite state machine on the chimp.
  - `_spin()` is called when the chimp is dizzy.
  - `_walk()` is called otherwise.

Demo: Punching the Chimpanzee

```python
    def _walk(self):
        """move the monkey across the screen, and turn at the
        newpos = self.rect.move((self.move, 0))
        # Change moving direction and flip the image if object
        if not self.area.contains(newpos):
            if self.rect.left < self.area.left \
                or self.rect.right > self.area.right:
                self.move = -self.move
                newpos = self.rect.move((self.move, 0))
                self.image = pg.transform.flip(
                    self.image, True, False)
        self.rect = newpos

    def _spin(self):
        """spin the monkey image"""
        center = self.rect.center
        self.dizzy = self.dizzy + 12
        if self.dizzy >= 360:
            self.dizzy = False
            self.image = self.original
        else:
            rotate = pg.transform.rotate
            self.image = rotate(self.original, self.dizzy)
        self.rect = self.image.get_rect(center=center)

    def punched(self):
        """this will cause the monkey to start spinning"""
        if not self.dizzy:
            self.dizzy = True
            self.original = self.image
```

The `_walk()` method:

- Walk towards the end of the screen normally.
- If the bounding box of the chimp object is outside the screen, the moving direction changes.
- The image is also flipped to be consistent with the moving direction.

The `_spin()` method:

- Spin the chimp with `pg.transform.rotate()`.
- Update the `image` and `rect` attribute.

The `punched()` method saves the original image and sets `dizzy` to True.

Demo: Punching the Chimpanzee

```python
# Create The Background
background = pg.Surface(screen.get_size())
background = background.convert()
background.fill((170, 238, 187))

# Put Text On The Background, Centered
if pg.font:
    font = pg.font.Font(None, 64)
    text = font.render("Pummel the Chimp!",
                       True, (10, 10, 10))
    textpos = text.get_rect(
        centerx=background.get_width()/2, y=10)
    background.blit(text, textpos)
```

Setup background color:

- Create a Surface for the background object with `pg.Surface()`.
- Convert the object to display on the screen with `Surface.convert()`.
- Setup the color with `Surface.fill()`.

Setup Text:

- Get a font with `pg.font.Font()`.
- Render a text with the font with `font.render()`.
- Set the position of the text to be the middle of the screen.
- Show text on the background with `blit()`.

Demo: Punching the Chimpanzee

```
screen.blit(background, (0, 0))
pg.display.update()
# Initialize Game Objects
whiff_sound = load_sound("whiff.wav")
punch_sound = load_sound("punch.wav")
chimp = Chimp()
fist = Fist()
allsprites = pg.sprite.RenderPlain((chimp, fist))
clock = pg.time.Clock()
```

- Show the background on the screen with `screen.blit()`.
- In Pygame, changes to the display interface are not immediately visible. A display must be updated in areas that have changed for them to be visible to the user.
- Call `pg.display.update()` to show the content to users.

- Load sounds and initialize objects.
- Organize the chimp and the fist object into a sprite group named `RenderPlain`. This sprite group can draw all the sprites it contains to the screen.
- Create a `clock` object to control the game's framerate.

Demo: Punching the Chimpanzee

```python
going = True
while going:
    # Set maximum frame per second
    clock.tick(60)
    # Handle Input Events
    for event in pg.event.get():
        if event.type == pg.QUIT: # quit
            going = False
        elif event.type == pg.KEYDOWN \
            and event.key == pg.K_ESCAPE: # es
            going = False
        elif event.type == pg.MOUSEBUTTONDOWN:
            if fist.punch(chimp):
                punch_sound.play() # punch
                chimp.punched()
            else:
                whiff_sound.play() # miss
        elif event.type == pg.MOUSEBUTTONUP:
            fist.unpunch()
    allsprites.update()
    # Draw Everything
    screen.blit(background, (0, 0))
    allsprites.draw(screen)
    pg.display.update()
pg.quit()
```

The event loop:

- Break the loop when press “Escape” or click the close button on the window.
- Punch the fist when click the mouse.
  - Call the `punch()` method of the `fist` object, which returns True when it collides with the `chimp` object.
  - Play the `punch_sound` and call the `punched()` method of the chimp object, if `puch()` returns True.
  - Otherwise, plays the whiff sound.
- `unpunch()` the fist when release the mouse button.
- Update all sprites and the display.

Demo: Punching the Chimpanzee

Run the demo!

Useful Links

- The pygame official site. There are many demo games.
  https://www.pygame.org/

- Other useful links for game engines.
  https://unity.com/
  https://www.unrealengine.com/

Appendix: Install and launch Python3

Installation of Python3

- Go to https://www.python.org
- Go to the "Download"
  Any Python version >= 3.7 is acceptable

(Python 2.x is not recommended, since it is not officially updated anymore)

- Download the installer depending your OS
  - 64 Bit Windows: Windows x86-64 executable installer
  - Mac OS: macOS 64-bit installer
  - 32 Bit Windows: Windows x86 executable installer

Launch Python3

- Check the version of your python3 by

```
> python3 --version
```

- Use your favourite editor (e.g. VS Code, PyCharm) to write a python program, say *testing.py*
- Type the following in *testing.py*:

```
print("Hello World!")
```

- Run the python program in your terminal

```
> python3 testing.py
```

```
Hello World!
```

Useful Links

- W3schools Python Tutorial
  - https://www.w3schools.com/python/python_intro.asp
- Anaconda3 is an alternative way to install python3, with convenient environment management.
  - https://www.anaconda.com
- Pycharm: a Python IDE for advanced developers
  - https://www.jetbrains.com/pycharm/
- Python in VS Code
  - https://code.visualstudio.com/docs/languages/python