**= 툴팁 소스 =**

class **ToolTip**(object):

def **\_\_init\_\_**(*self*,widget):

*self*.widget=widget

*self*.tip\_window=None

def **show\_tip**(*self*,tip\_text):

if *self*.tip\_window or not tip\_text:

return

x,y,\_cx,cy=*self*.widget.bbox(*"insert"*) # 바운딩 박스,cx,cy는 폭과 높이, 문자를 표시하지 않으면 none을 반환

x=x+*self*.widget.winfo\_rootx()+25 # 부모 윈도우를 기준으로 해서 왼쪽의 x좌표와 윗면의 y좌표를 반환한다.

y=y+cy+*self*.widget.winfo\_rooty()+25

*self*.tip\_window=tw=tk.Toplevel(*self*.widget)

tw.wm\_overrideredirect(1) #데코레이션을 해주냐, 안해주냐 예를 들어 확채,축소,x그런게 다 데코레이션이다.

tw.wm\_geometry(*"+%d+%d"*%(x,y)) #윈도우가 놓일 위치를 제어한다.

label=tk.Label(tw,text=tip\_text,justify=tk.LEFT,background=*"#ffffe0"*,relief=tk.GROOVE,borderwidth=1,font=(*"tahomg"*,*"8"*,*"normal"*))

label.pack(ipadx=1)

def **hide\_tip**(*self*):

tw=*self*.tip\_window

*self*.tip\_window=None

if tw:

tw.destroy()

def **create\_ToolTip**(widget, text):

toolTip=ToolTip(widget)

def **enter**(event):

toolTip.show\_tip(text)

def **leave**(event):

toolTip.hide\_tip()

widget.bind(*"<Enter>"*,enter)

widget.bind(*"<Leave>"*,leave)

spin=tk.Spinbox(mighty,values=(1,2,4,42,100),width=5,bd=8,command=\_spin)

spin.grid(column=0,row=2)

create\_ToolTip(spin,*'This is a spincontrol'*)

**= 툴팁 설명**(매우 상세함) **=**

툴팁은 사용자에게 위젯의 기능이나 사용법을 알려주기 위한 위젯이다.

우리는 create\_ToolTip(위젯,텍스트) 소스를 통해 툴팁을 생성할 수 있다.

첫 번째 인수인 '위젯'은 툴팁을 어떤 위젯에 나타낼지를 알려주는 것이고 두 번째 인수인 '텍스트'는 툴팁에 적을 데이터가 무엇인지 알려주는 것이다.

이렇게 '위젯','스트링'을 인수로 보내서 create\_ToolTip함수를 호출하는게 먼저 컴파일된다.

컴파일된 create\_ToolTip함수를 살펴보겠다.

이 함수에서는 시작부터 Class Tooltip을 인스턴스화한다.

툴팁 클래스를 인스턴스화시켰을 때, '위젯(툴팁을 나타낼 위젯)'이 전달되고 생성자가 바로 호출된다.

생성자에서는 '위젯'을 저장한 self.widget변수를 하나 만들고, 아무 것도 넣지 않은 self.tip\_window변수를 만들었다.

(self가 변수 앞에 붙으면 self가 객체를 자기 자신으로 나타내서 self.widget이 tooltip.widget 대신 사용할 수 있다. 뿐만 아니라 이 툴팁 객체 안의 모든 메소드에서 생성한 (self.변수)를

변수 값 그대로 이용할 수 있다.)

이후 다시 create\_ToolTip함수로 돌아온다. 돌아와서는 bind메소드를 이용해서 마우스 포인터가 '위젯'에 있을 때는 enter함수를 호출하게 하고

마우스 포인터가 '위젯'에 없을 때는 leave함수를 호출하도록 하였다.

마우스 포인터가 '위젯'에 있다고 가정하고 enter함수를 살펴보겠다.

이 함수는 호출되자마자 툴팁 인스턴스에 있는 show\_tip메소드를 호출함과 동시에 '텍스트(툴팁에 적을 데이터)'를 전달한다.

그럼 호출된 show\_tip메소드를 살펴보겠다.

만일(self.tip\_window가 존재하거나) 혹은 (self.tip\_window가 존재하지 않고 tip\_text도 존재하지 않을 때) 리턴한다. 생성자에서

self.tip\_window는 아무 것도 없다고 정의됐고 tip\_text는 인수로 받은 (툴팁에 적을 데이터)이다.

즉 self.tip\_window가 존재하지 않고 tip\_text도 존재하니까 일단은 리턴되지 않고 if문을 빠져 나온다.

이후 bbox라고 하는 메소드를 이용해서 x,y,cx,cy라는 변수를 지정하였다.

x,y,cx,cy는 각각 x좌표,y좌표,폭과 높이를 나타내는 변수고 초기값은 임의로 지정되었다.

x는 winfo\_rootx라고 하는 메소드가 이용되서 부모 윈도우 왼쪽 틀의 x좌표를 반환 후 그 값에 x+200을 더해서 저장된다.

y는 winfo\_rooty라고 하는 메소드가 이용되서 부모 윈도우 위쪽 틀의 y좌표를 반환 후 그 값에 y+cy+70을 더해서 저장된다.

이후 Toplevel생성자를 이용해서 임의의 윈도우 창을 하나 만드는데, 이 윈도우 창은 '위젯(툴팁을 나타낼 위젯)'안에 생성된다.

그리고 이 임의의 윈도우를 tw변수에 저장하고 self.tip\_window변수에도 저장한다. 이 소스가 컴파일된 이후부터 tw와 self.tip\_window는 임의의 윈도우가 되겠다.

그리고 tw에 wm\_overrideredirect라고 하는 메소드를 이용했다. 이 메소드는 데코레이션(x버튼,확대,축소 버튼)을 숨기냐,안숨기냐를 결정하는 메소드이다. 즉 괄호 안에 True가 왔으니

tw(임의의 윈도우창)의 데코레이션을 숨기게 된다.

이후 tw에 wm\_geometry 라고 하는 메소드를 이용했다. 이 메소드는 위치를 제어하는 메소드이다. 즉 괄호안에 좌표를 넣으면 그 좌표에 맞추어서 tw(임의의 윈도우창) 위치를 결정된다.

이후 tw(임의의 윈도우창)안에 라벨을 만들었다. 라벨은 텍스트는 (툴팁에 적을 데이터)로 설정했고 왼쪽에 있게 하였으며 배경색은 lightyellow색을 주었다. 테두리 옵션은 groove를 줘서 왼쪽과 위쪽 테두리만

검정색이 되게 하였고 폰트는 "tahomg"라는 것을 주었다. 이후 pack메소드와 ipadx속성을 이용해서 텍스트 좌우 여백에 1만큼을 주었다.

결국 툴팁은 toplevel을 이용한 (임의의 윈도우창)을 디자인해서 잘 배치한 것이라고 봐도 된다.

이 모든 소스가 작성되면, 위젯에 마우스 포인터가 있을 때 툴팁이 나오는 것이다.

이번엔 마우스 포인터가 처음부터 '위젯'에 없다고 가정하고 leave함수를 살펴보겠다.

이 함수는 호출되자마자 툴팁 인스턴스에 있는 hide\_tip메소드를 호출하기만 한다.

hide\_tip을 살펴보겠다.

처음부터 포인터가 '위젯'에 없다고 가정했으니, 아직 show\_tip메소드는 호출되지 않았을 것이고 self.tip\_window변수는 아무 값도 가지지 않게 된다. 이 None을 tw변수에 넣어서 생성했고

만일 tw값이 존재하면 tw자체를 없애게 만들었다. 하지만 None이니 if문 내용이 실행은 안되겠다.

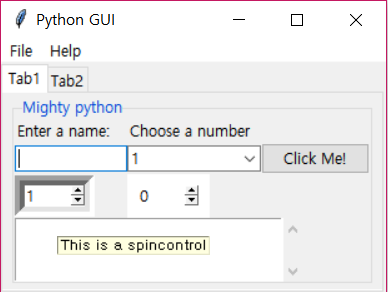
그럼 처음에는 마우스 포인터가 '위젯'에 있었지만 없어졌다고 가정해보자. 이 때는 show\_tip메소드가 이미 호출되었을 것이고 self.tip\_window변수는 툴팁(임의의 윈도우)를 가지게 된다.

이 윈도우를 tw에 넣고 만일 tw값이 존재하면 tw자체를 없애게 만들었다. 이 때는 툴팁(임의의 윈도우)를 없애게 된다. 그리고 self.tip\_window변수는 다시 None으로 만들었다.

마지막으로 마우스 포인터가 '위젯'에 없었지만 있게 됐다고 가정해보다. 이 때는 hide\_tip메소드가 이미 호출되었을 것이고 self.tip\_window변수는 None을 가진다. 즉 show\_tip메소드에서

(tip\_text가 None이 아닌 이상) 리턴될 일은 전혀 없다. 포인터가 처음부터 위젯에 있었던 경우, 포인터가 처음에는 위젯에 없었지만 있게 된 경우 모두 self.tip\_window변수가 None을 갖게 되기 때문이다.

**= 툴팁 결과 =**



**= 프로그래스바 소스 =**

def **run\_progressbar**():

progress\_bar[*'maximum'*]=100

for i in range(101):

sleep(0.05)

progress\_bar[*'value'*]=i

progress\_bar.update()

progress\_bar[*'value'*]=0

def **start\_progressbar**():

progress\_bar.start()

def **stop\_progressbar**():

progress\_bar.stop()

def **progressbar\_stop\_after**(wait\_ms=1000):

win.after(wait\_ms,progress\_bar.stop)

ttk.Button(buttons\_frame,text=*"Run Progressbar"*,command=run\_progressbar).grid(column=0,row=0,sticky=*'W'*)

ttk.Button(buttons\_frame,text=*"Start Progressbar"*,command=start\_progressbar).grid(column=0,row=1,sticky=*'W'*)

ttk.Button(buttons\_frame,text=*"Stop Progressbar"*,command=stop\_progressbar).grid(column=0,row=2,sticky=*'W'*)

ttk.Button(buttons\_frame,text=*"Stop after second"*,command=lambda:progressbar\_stop\_after(2000)).grid(column=0,row=3,sticky=*'W'*)

progress\_bar=ttk.Progressbar(tab2,orient=*'horizontal'*,length=286,mode=*'determinate'*)

progress\_bar.grid(column=0,row=3,pady=2)

**= 프로그래스바 설명** **=**

프로그래스 바 위젯은 현재 퍼센트를 그래픽으로 나타내거나 진행하고 있다는 것을 보여주는 위젯이다.

흔히 프로그램을 설치할 때, 게임이 몇 퍼센트나 다운로드 되었는지 나타내는 진행 바를 예시로 들 수 있겠다.

이 위젯은 ttk모듈에 있는 progressbar생성자를 이용해서 만들 수 있다. 이 위젯은 탭2에 만들었고 수평으로 만들었으며 길이는 286px, 모드는 determinate로 하였다.

모드에는 determinate와 indeterminate가 있는데 determinate는 현재의 퍼센트를 그래픽으로 나타내주고, indeterminate는 현재 진행하고 있다는 것을 그래픽으로 보여주기만 한다.

start\_progressbar함수는 progressbar클래스에 있는 start메소드를 이용해서 프로그래스바가 동작하게 만드는 함수이다.

stop\_progressbar함수는 progressbar클래스에 있는 stop메소드를 이용해서 프로그래스바가 멈추게 만드는 함수이다. 단 start메소드에 의해 동작하고 있는 프로그래스바만 멈추게 할 수 있다.

run\_progressbar함수는 사용자에 의해 프로그래스바가 동작하게 만드는 함수이다. 이 함수의 내용은 아래와 같다.

먼저 프로그래스바의 최대치를 100으로 설정한다. 그리고 i를 0부터 100까지 반복시킨다. 반복문에서는 (1. time모듈의 sleep함수를 통해서 0..05초를 기다린다. 2. 프로그래스바의 현재 값을 i로 두고

i에 따라 그래픽으로 나타낸다. 3, 프로그래스바의 그래픽 수치를 바꿀 준비를 한다.) 가 계속 진행되며 반복이 끝날 때는 프로그래스바도 꽉 차있는 상태가 된다.

그리고 반복을 빠져 나오면 다시 프로그래스바 값을 0으로 두고 그래픽을 바꾸었다.

progressbar\_stop\_after함수는 win이 참조하는 Tk인스턴스에 있는 after메소드를 이용해서 진행된다.

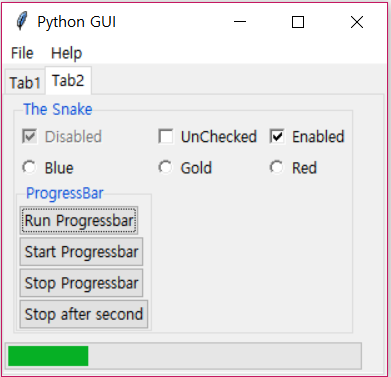
첫 번째 인수는 (초)값을 가지는 변수이고 두 번째 인수는 제어할 위젯이고 세 번째 인수는 호출할 함수이다.

즉 이것을 합쳐서 말하면 제어할 위젯을 정하고 인수 값만큼 초를 지나면 함수가 호출되서 위젯에 영향을 받게 한다는 의미이다.

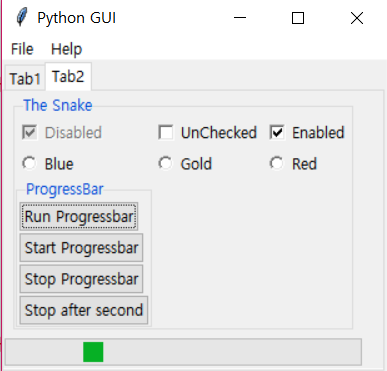
위 소스에서는 프로그래스바를 제어할 위젯으로 정하고 1초 후에 stop함수를 호출해서 멈추게 한다는 의미를 갖게 된다.

lambda:progressbar\_stop\_after(2000)는 콜론 뒤에 있는 함수를 호출하고 인수 값을 전달한 모습이다. 람다는 이렇게 인수 값을 전달할 때 유용하다.

**= 프로그래스바 결과 =**



-determinate 모드-



-indeterminate 모드-